Model Checking Real-Time Systems

Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, Joël Ouaknine, James Worrell

▶ **To cite this version:**

Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, Joël Ouaknine, et al.. Model Checking Real-Time Systems. Handbook of model checking, Springer-Verlag, pp.1001-1046, 2018, 978-3-319-10574-1. 10.1007/978-3-319-10575-8_29 . hal-01889280

HAL Id: hal-01889280

https://hal.archives-ouvertes.fr/hal-01889280

Submitted on 8 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Model Checking Real-Time Systems

Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey,
Joël Ouaknine, and James Worrell

Abstract This chapter surveys timed automata as a formalism for model checking real-time systems. We begin with introducing the model, as an extension of finite-state automata with real-valued variables for measuring time. We then present the main model-checking results in this framework, and give a hint about some recent extensions (namely weighted timed automata and timed games).

1 Introduction

Timed automata were introduced by Rajeev Alur and David Dill in the early 1990s [13] as finite-state automata equipped with real-valued variables for measuring time between transitions in the automaton. These variables all evolve at the same rate; they can be reset along some transitions, and used as *guards* along other transitions or invariants to be preserved while letting time elapse in locations of the automaton.

Patricia Bouyer
LSV – CNRS & ENS Cachan, U. Paris-Saclay, France, e-mail: patricia.bouyer@lsv.fr.

Uli Fahrenberg
IRISA – INRIA & U. Rennes I, France, e-mail: uli.fahrenberg@irisa.fr.

Kim G. Larsen
Computer Science Department – U. Aalborg, Denmark, e-mail: kgl@cs.aau.dk.

Nicolas Markey
LSV – CNRS & ENS Cachan, U. Paris-Saclay, France, e-mail: nicolas.markey@lsv.fr.

Joël Ouaknine
Computer Science Department – U. Oxford, UK, e-mail: joel.ouaknine@cs.ox.ac.uk.

James Worrell
Computer Science Department – U. Oxford, UK, e-mail: james.worrell@cs.ox.ac.uk.

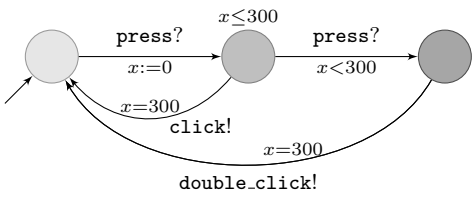


Fig. 1 A timed automaton

Timed automata have proven very convenient for modeling and reasoning about real-time systems: they combine a powerful formalism with advanced expressiveness and efficient algorithmic and tool support, and have become a model of choice in the framework of verification of embedded systems. The timed-automata formalism is now routinely applied to the analysis of real-time control programs [85, 127] and timing analysis of software and asynchronous circuits [60, 146, 150]. Similarly, numerous real-time communication protocols have been analysed using timed automata technology, often with inconsistencies being revealed [95, 144]. During the last few years, timed-automata-based schedulability and response-time analysis of multitasking applications running under real-time operating systems have received substantial research effort [61, 78, 80, 109, 156]. Also, for optimal planning and scheduling, (priced) timed automata technology has been shown to provide competitive and complementary performances with respect to classical approaches [1, 2, 32, 86, 97, 108, 119]. Finally, controller synthesis from timed games has been applied to a number of industrial case studies [6, 71, 110].

The handiness of this formalism is exemplified in Fig. 1, modeling a (simplified) computer mouse: this automaton receives `press` events, corresponding to an action of the user on the button of the mouse. When two such events are close enough (less than 300 milliseconds apart), this is translated into a `double_click` event.

Because clock variables are real-valued, timed automata are in fact infinite-state models, where a configuration is given by a location of the automaton and a valuation of the clocks. Timed automata have two kinds of transitions: *action transitions* correspond to firing a transition of the automaton, and *delay transitions* correspond to letting time elapse in the current location of the automaton. Section 2 provides the definitions of this framework. The main technical ingredient for dealing with this infinity of states is the *region abstraction*, as we explain in Sect. 3. Roughly, two clock valuations are called *region equivalent* whenever they satisfy the exact same set of constraints of the form $x - y \bowtie c$, where the difference of two clocks x and y is compared to some integer c (no greater than some constant M). This abstraction can be used to develop various algorithms, in particular for deciding bisimilarity (Sect. 4) or model checking some quantitative extensions of the classical temporal logics CTL and LTL (Sect. 6). We also show that some problems are undecidable, most notably language containment (Sect. 5) and model checking the full quantitative extension of LTL. On the practical side, regions are in some sense too fine-grained, and another abstraction, called *zones*, is preferred for implementation purposes. Roughly, zones provide a way of grouping many

regions together, which is often relevant in practical situations. We explain in Sect. 7 how properties of timed automata can be verified in practice.

Finally, we conclude this chapter with two powerful extensions of timed automata: first, *weighted timed automata* allow for modeling quantitative constraints beyond time; since resource (e.g., energy) consumption is usually tightly bound to time elapsing, timed automata provide a convenient framework for modeling such quantitative aspects of systems. Unlike hybrid systems (see Chap. 28hybrid), weighted timed automata still enjoy some nice decidability properties (in restricted settings though), as we explain in Sect. 8. Then in Sect. 9 we present *timed games*, which are very powerful and convenient for dealing with the controller synthesis problem (see Chap. 25gamesynth) in a timed framework. Timed games also provide an interesting way of modeling uncertainty in real-time systems, assuming worst-case resolution of the uncertainty while still trying to benefit from non-worst-case situations.

2 Timed Automata

In this chapter, we consider as time domain the set $\mathbb{R}_{\geq 0}$ of non-negative reals. While discrete time might look reasonable for representing digital systems, it assumes synchronous interactions between the systems. We refer to [17, 27, 67, 102] for more discussions on this point.

Let Σ be a finite set of *actions*. A *time sequence* is a finite or infinite non-decreasing sequence of non-negative reals. A *timed word* is a finite or infinite sequence of pairs $(a_1, t_1) \dots (a_p, t_p) \dots$ such that $a_i \in \Sigma$ for every i, and $(t_i)_{i \geq 1}$ is a time sequence. An infinite timed word is *converging* if its time sequence is bounded above (or, equivalently, converges).

We consider a finite set C of variables, called *clocks*. A *(clock) valuation* over C is a mapping $v \colon C \to \mathbb{R}_{\geq 0}$ which assigns to each clock a real value. The set of all clock valuations over C is denoted $\mathbb{R}_{\geq 0}^C$, and $\mathbf{0}_C$ denotes the valuation assigning 0 to every clock $x \in C$.

Let $v \in \mathbb{R}_{\geq 0}^C$ be a valuation and $t \in \mathbb{R}_{\geq 0}$; the valuation $v + t$ is defined by $(v + t)(x) = v(x) + t$ for every $x \in C$. For a subset r of C, we denote by $v[r]$ the valuation obtained from v by resetting clocks in r; formally, for every $x \in r$, $v[r](x) = 0$ and for every $x \in C \setminus r$, $v[r](x) = v(x)$.

The set $\Phi(C)$ of *clock constraints* over C is defined by the grammar

$$\Phi(C) \ni \varphi ::= x \bowtie k \ \mid \ \varphi_1 \wedge \varphi_2 \ (x \in C,\ k \in \mathbb{Z} \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}).$$

We will sometimes make use of *diagonal clock constraints*, which additionally allow constraints of the form $x - y \bowtie k$. We write $\Phi_d(C)$ for the extension of $\Phi(C)$ with diagonal constraints. If $v \in \mathbb{R}_{\geq 0}^C$ is a clock valuation, we write $v \models \varphi$ when v satisfies the clock constraint φ, and we say that v satisfies $x \bowtie k$ whenever $v(x) \bowtie k$ (similarly, v satisfies $x - y \bowtie k$ when $v(x) - v(y) \bowtie k$).

If φ is a clock constraint, we write $[\![\varphi]\!]_C$ for the set of clock valuations $\{v \in \mathbb{R}^C_{\geq 0} \mid v \models \varphi\}$.

Definition 1 ([13]). A *timed automaton* is a tuple $(L, \ell_0, C, \Sigma, I, E)$ consisting of a finite set L of locations with initial location $\ell_0 \in L$, a finite set C of clocks, an invariant[1] mapping $I\colon L \to \Phi(C)$, a finite alphabet Σ and a set $E \subseteq L \times \Phi(C) \times \Sigma \times 2^C \times L$ of edges. We shall write $\ell \xrightarrow{\varphi,a,r} \ell'$ for an edge $(\ell, \varphi, a, r, \ell') \in E$; formula φ is the *guard* of the transition (and has to be satisfied when the transition is taken), and r is the set of clocks that are set to zero after taking that transition.

Later for defining languages accepted by timed automata we may add final or repeated (Büchi) locations, and for defining logical satisfaction relations we may add atomic proposition labeling to timed automata. However for readability reasons we omit them here.

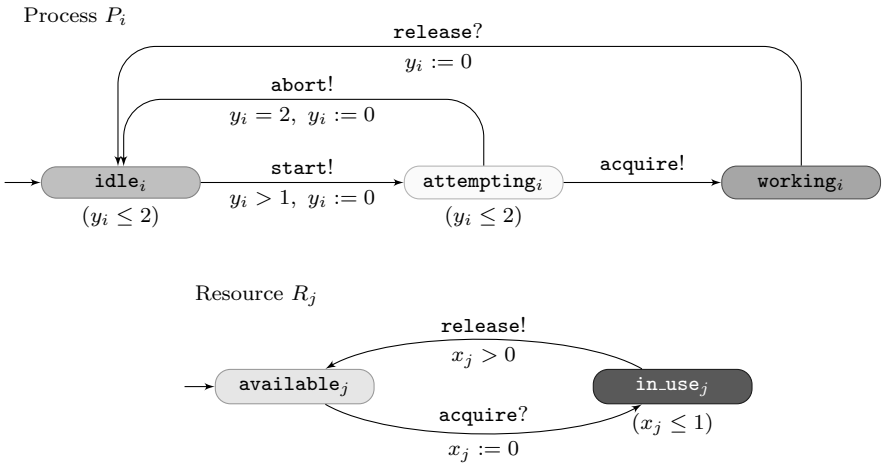


Fig. 2 Model of a process that acquires and releases two resources. Here and in the rest of this chapter, transitions are decorated with their associated guards (e.g., $x_j > 0$), letters of the alphabet (e.g., `release`!), and resets (written e.g., as $x_j := 0$); invariants (if any) are written in brackets below their corresponding locations

Example 1. Figure 2 shows timed automata models for processes and resources. Processes can use resources, but mutual exclusion is expected. The model for process P_i is given in the upper part of the figure, whereas the model for resource R_j is given in the lower part of the figure. Starting in the `idle` location, the process should start within one to two time units requesting a

[1] The original definition of timed automata [13] did not contain invariants in locations, but had Büchi conditions to enforce liveness. Invariants were added by [103]. Several other convenient extensions have been introduced since then, which we discuss in Sect. 3.2.

resource. After two time units it must abort its request, unless before two time units it acquires the resource and goes to the `working` location. The resource is released when the process is done working with it.

Our model for a resource has two locations, and when the resource is available, it can be acquired and should be released within one time unit.

The *operational semantics* of a timed automaton $A = (L, \ell_0, C, \Sigma, I, E)$ is the (infinite-state) timed transition system $[\![A]\!] = (S, s_0, \mathbb{R}_{\geq 0} \times \Sigma, T)$ given as follows:

$$S = \big\{(\ell, v) \in L \times \mathbb{R}_{\geq 0}^{C} \;\big|\; v \models I(\ell)\big\} \qquad s_0 = (\ell_0, \mathbf{0}_C)$$
$$T = \big\{(\ell, v) \xrightarrow{d,a} (\ell', v') \;\big|\; \forall d' \in [0, d] : v + d' \models I(\ell),$$
$$\text{and } \exists \ell \xrightarrow{\varphi,a,r} \ell' \in E : \; v + d \models \varphi, \text{ and } v' = (v + d)[r]\big\}$$

In words, one can jump from one state (ℓ, v) to another one (ℓ', v') by selecting a delay to be elapsed in ℓ (provided that the invariant of location ℓ is fulfilled in the meantime) and an edge of the automaton, which is taken after the delay, provided that its guard is satisfied at that time. In this semantics, a transition combines both a delay and (followed by) the application of an edge of the automaton. A slightly different semantics is sometimes used, which distinguishes pure-delay transitions (denoted $\xrightarrow{d}$, for $d \in \mathbb{R}_{\geq 0}$) and pure-action transitions (denoted $\xrightarrow{a}$ with $a \in \Sigma$).

A (finite or infinite) *run* of a timed automaton A is a (finite or infinite) path $\rho = (\ell_0, v_0) \xrightarrow{d_1,a_1} (\ell_1, v_1) \xrightarrow{d_2,a_2} \cdots$ in the transition system $[\![A]\!]$, which starts with $v_0 = \mathbf{0}_C$. Given a run $\rho = (\ell_0, v_0) \xrightarrow{d_1,a_1} (\ell_1, v_1) \xrightarrow{d_2,a_2} (\ell_2, v_2) \ldots$, we say that it reads the timed word $w = (a_1, t_1)(a_2, t_2) \ldots$ where for every i, $t_i = \sum_{j \leq i} d_j$. A run is *time-divergent* if its time sequence $(t_i)_i$ diverges. A timed automaton is *non-Zeno* if any finite run can be extended into a time-divergent run.

Example 2. The process R_1 given in Fig. 2 has a single clock x_1, and has as set of states $S = \{\texttt{available}_1\} \times \mathbb{R}_{\geq 0} \cup \{\texttt{in\_use}_1\} \times [0, 1]$ where we identify valuations (for the single clock x_1) with the value of x_1. We give below a possible run for the resource R_1:

$$(\texttt{available}_1, 0) \xrightarrow{5.4,\texttt{acquire?}} (\texttt{in\_use}_1, 0) \xrightarrow{0.8,\texttt{release!}} (\texttt{available}_1, 0.8) \xrightarrow{1.4,\texttt{acquire?}} (\texttt{in\_use}_1, 0) \to \ldots$$

In location `in_use`, the invariant is satisfied in this run because the value of x_1 never exceeds 0.8 (hence satisfies the constraint $x_1 \leq 1$).

We now define the parallel composition of timed automata, which allows us to define systems in a compositional way [23, 107]. Let $(A_i)_{1 \leq i \leq n}$ be n

timed automata, where $A_i = (L_i, \ell_0^i, C_i, \Sigma_i, I_i, E_i)$. Assume that all Σ_i's are disjoint, and all C_i's are disjoint. If Σ is a new alphabet, given a (partial) synchronization function $f\colon \prod_{i=1}^n (\Sigma_i \cup \{-\}) \to \Sigma$, the *synchronized product* (or *parallel composition*) $(A_1 \parallel A_2 \parallel \cdots \parallel A_n)_f$ is the timed automaton $A = (L, \ell_0, C, \Sigma, I, E)$ where $L = L_1 \times \cdots \times L_n$, $\ell_0 = (\ell_0^1, \ldots, \ell_0^n)$, $C = C_1 \cup \cdots \cup C_n$, $I((\ell_1, \ldots, \ell_n)) = \bigwedge_{i=1}^n I_i(\ell_i)$ for every $(\ell_1, \ldots, \ell_n) \in L_1 \times \cdots \times L_n$, and the set E is composed of the transitions $(\ell_1, \ldots, \ell_n) \xrightarrow{\varphi, a, r} (\ell'_1, \ldots, \ell'_n)$ whenever

1. there exists $(\alpha_1, \ldots, \alpha_n) \in \prod_{i=1}^n (\Sigma_i \cup \{-\})$ such that $f(\alpha_1, \ldots, \alpha_n) = a$;
2. if $\alpha_i = -$, then $\ell'_i = \ell_i$;
3. if $\alpha_i \neq -$, then there is a transition $\ell_i \xrightarrow{\varphi_i, \alpha_i, r_i} \ell'_i$ in E_i
4. $\varphi = \bigwedge \{\varphi_i \mid \alpha_i \neq -\}$ and $r = \bigcup \{r_i \mid \alpha_i \neq -\}$

Example 3. We build on the system given in Fig. 2. The process and the resources are not expected to run independently, but they are part of a global system where the process should synchronize with the resources. Hence for this system we have a natural synchronization function f defined by Table 1.

P_1	P_2	R_1		
start!	$-$	$-$	$\to$	$\texttt{start}_1$
$-$	start!	$-$	$\to$	$\texttt{start}_2$
abort!	$-$	$-$	$\to$	$\texttt{abort}_1$
$-$	abort!	$-$	$\to$	$\texttt{abort}_2$
acquire!	$-$	acquire?	$\to$	$\texttt{acquire}_1$
$-$	acquire!	acquire?	$\to$	$\texttt{acquire}_2$
release?	$-$	release!	$\to$	$\texttt{release}_1$
$-$	release?	release!	$\to$	$\texttt{release}_2$

Table 1 The synchronization function f

The global system $(P_1 \parallel P_2 \parallel R_1)_f$ (more precisely the part which is reachable from the initial state) is depicted in Fig. 3. This automaton is rather complex, and the component-based definition as $(P_1 \parallel P_2 \parallel R_1)_f$ is much easier to understand. Furthermore this allows addition of other processes and other resources to the system without any effort.

3 Checking Reachability

In this section we are interested in the most basic problem regarding timed automata, namely reachability. This problem asks, given a timed automaton A, whether a distinguished set of locations F of A is reachable or not.

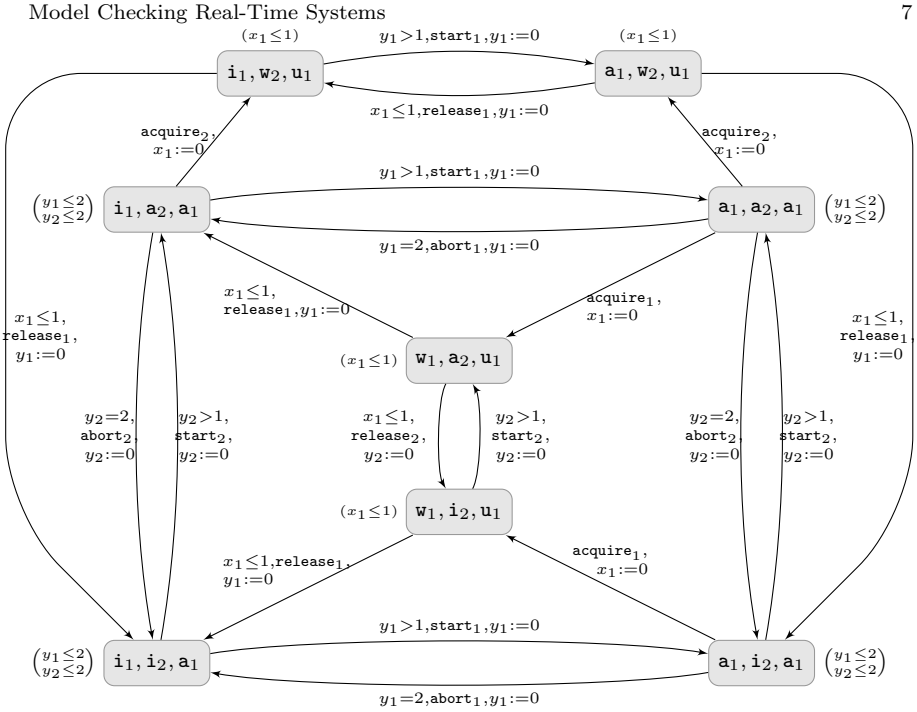

$(x_1 \le 1)$
$y_1 > 1, \mathtt{start}_1, y_1 := 0$
$(x_1 \le 1)$
$\mathtt{i}_1, \mathtt{w}_2, \mathtt{u}_1$
$\mathtt{a}_1, \mathtt{w}_2, \mathtt{u}_1$
$x_1 \le 1, \mathtt{release}_1, y_1 := 0$
$\mathtt{acquire}_2, x_1 := 0$
$\mathtt{acquire}_2, x_1 := 0$
$y_1 > 1, \mathtt{start}_1, y_1 := 0$
$\binom{y_1 \le 2}{y_2 \le 2}$
$\mathtt{i}_1, \mathtt{a}_2, \mathtt{a}_1$
$\mathtt{a}_1, \mathtt{a}_2, \mathtt{a}_1$
$\binom{y_1 \le 2}{y_2 \le 2}$
$y_1 = 2, \mathtt{abort}_1, y_1 := 0$
$x_1 \le 1, \mathtt{release}_1, y_1 := 0$
$\mathtt{acquire}_1, x_1 := 0$
$x_1 \le 1, \mathtt{release}_1, y_1 := 0$
$x_1 \le 1, \mathtt{release}_1, y_1 := 0$
$(x_1 \le 1)$
$\mathtt{w}_1, \mathtt{a}_2, \mathtt{u}_1$
$y_2 = 2, \mathtt{abort}_2, y_2 := 0$
$y_2 > 1, \mathtt{start}_2, y_2 := 0$
$x_1 \le 1, \mathtt{release}_2, y_2 := 0$
$y_2 > 1, \mathtt{start}_2, y_2 := 0$
$y_2 = 2, \mathtt{abort}_2, y_2 := 0$
$y_2 > 1, \mathtt{start}_2, y_2 := 0$
$(x_1 \le 1)$
$\mathtt{w}_1, \mathtt{i}_2, \mathtt{u}_1$
$x_1 \le 1, \mathtt{release}_1, y_1 := 0$
$\mathtt{acquire}_1, x_1 := 0$
$y_1 > 1, \mathtt{start}_1, y_1 := 0$
$\binom{y_1 \le 2}{y_2 \le 2}$
$\mathtt{i}_1, \mathtt{i}_2, \mathtt{a}_1$
$\mathtt{a}_1, \mathtt{i}_2, \mathtt{a}_1$
$\binom{y_1 \le 2}{y_2 \le 2}$
$y_1 = 2, \mathtt{abort}_1, y_1 := 0$

Fig. 3 The global system $(P_1 \parallel P_2 \parallel R_1)_f$, where "$\mathtt{i}_j$" (resp. "$\mathtt{a}_j$", "$\mathtt{w}_j$") stands for location "$\mathtt{idle}_j$" (resp. "$\mathtt{attempting}_j$", "$\mathtt{working}_j$") in P_j, and "$\mathtt{a}_1$" (resp. "$\mathtt{u}_1$") stands for location "$\mathtt{available}_1$" (resp. "$\mathtt{in\_use}_1$") in R_1

3.1 Region Equivalence

For the rest of this section we fix a timed automaton $A = (L, \ell_0, C, \Sigma, I, E)$ and a set of target locations F. For every clock $x \in C$ we let M_x be the maximal constant clock x is compared to in A.

Two valuations $v, v' \colon C \to \mathbb{R}_{\ge 0}$ are said to be *region equivalent w.r.t. maximal constants* $M = (M_x)_{x \in C}$, denoted $v \cong_M v'$, if [2]

- for all $x \in C$, $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ or $v(x), v'(x) > M_x$, and
- for all $x \in C$ with $v(x) \le M_x$, $\langle v(x) \rangle = 0$ iff $\langle v'(x) \rangle = 0$, and
- for all $x, y \in C$ with $v(x) \le M_x$ and $v(y) \le M_y$, $\langle v(x) \rangle \le \langle v(y) \rangle$ iff $\langle v'(x) \rangle \le \langle v'(y) \rangle$.

The equivalence classes of valuations with respect to $\cong_M$ are called *regions* (with maximal constants M). The number of regions is finite and is bounded above by $n! \cdot 2^n \cdot \prod_{x \in C}(2M_x + 2)$. Region equivalence of valuations is extended to states of A by declaring that $(\ell, v) \cong_M (\ell', v')$ whenever $\ell = \ell'$ and $v \cong_M v'$. We write $[\ell, v]_{\cong_M}$ for the equivalence class of (ℓ, v).

[2] For $d \in \mathbb{R}_{\ge 0}$ we write $\lfloor d \rfloor$ and $\langle d \rangle$ for the integral and fractional parts of d, i.e., $d = \lfloor d \rfloor + \langle d \rangle$.

Region equivalence enjoys nice properties, the most important of which is that it is a *time-abstracted bisimulation* in the following sense:

Definition 2. A relation R on the states of A is a *time-abstracted bisimulation* if $(\ell_1, v_1)R(\ell_2, v_2)$ and $(\ell_1, v_1) \xrightarrow{d_1,a} (\ell_1', v_1')$ for some $d_1 \in \mathbb{R}_{\geq 0}$ and $a \in \Sigma$ imply $(\ell_2, v_2) \xrightarrow{d_2,a} (\ell_2', v_2')$ for some $d_2 \in \mathbb{R}_{\geq 0}$, with $(\ell_1', v_1')R(\ell_2', v_2')$ and vice versa.

In other words, from two equivalent states, the automaton can take the same transitions, except that the values of the delays might have to be changed. This fundamental property has important consequences, like the construction of an interesting finite abstraction for A.

Definition 3. The *region automaton* $\mathcal{R}_{\cong_M}(A) = (S, s_0, \Sigma, T)$ associated with A has as set of states the quotient $S = \left(L \times \mathbb{R}^C_{\geq 0}\right)_{/\cong_M}$, as initial state $s_0 = [\ell_0, \mathbf{0}_C]_{\cong_M}$, and as transitions all the $[\ell, v]_{\cong_M} \xrightarrow{a} [\ell', v']_{\cong_M}$ for which $(\ell, v) \xrightarrow{d,a} (\ell', v')$ for some $d \in \mathbb{R}_{\geq 0}$. The target set of $\mathcal{R}_{\cong_M}(A)$ is defined as $S_F = \{[\ell, v]_{\cong_M} \mid \ell \in F\}$.

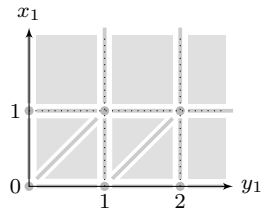


Fig. 4 Clock regions for the system $(P_1 \parallel R_1)_{f_1}$

The region automaton $\mathcal{R}_{\cong_M}(A)$ is a finite automaton whose size is exponential compared with the size of A. It can be used to check, e.g., reachability properties (or equivalently language emptiness):

Proposition 1. *The set of locations F is reachable in A from ℓ_0 iff S_F is reachable in $\mathcal{R}_{\cong_M}(A)$ from s_0.*

The region automaton has exponentially larger size, but checking a reachability property can be done on the fly, hence this can be done in polynomial space. One of the most fundamental theorems in the model checking of timed automata can be stated as follows.

Theorem 1 ([13]). *The reachability problem in timed automata is* PSPACE*-complete.*

Example 4. Restricting our running example to process P_1 and resource R_1 (we assume f_1 is the synchronization function f restricted to those two processes), the global system has two clocks, x_1 and y_1. The set of regions is then depicted in Fig. 4. There are 28 regions. The (reachable part of the) corresponding region automaton is depicted in Fig. 5. In this drawing we omit indices over names of locations since they should all be 1; also, the thick "release" transition at the top corresponds to a set of transitions from all the states on the right to all the states on the left.

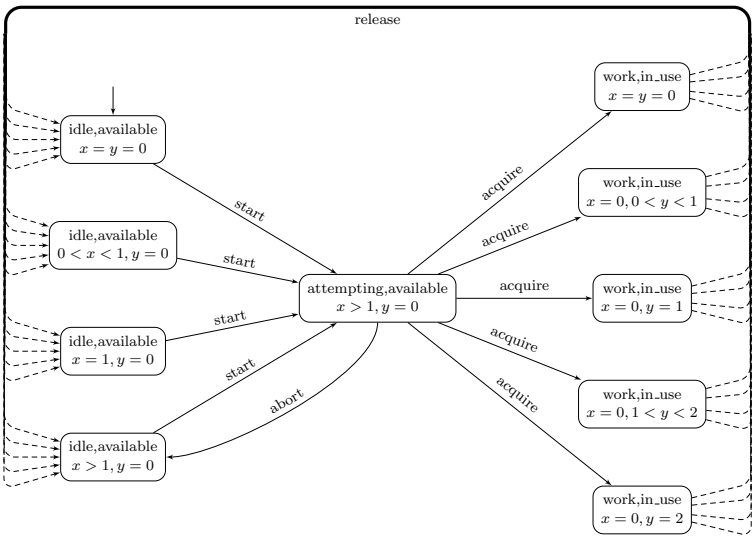


Fig. 5 Region automaton for $(P_1 \parallel R_1)_{f_1}$

3.2 Some Extensions of Timed Automata

Timed automata are the most basic model for representing systems with (quantitative) real-time constraints. It is natural to extend the model with features that help in modeling real systems. However decidability of the reachability problem remains the fundamental property one wants to preserve. In this subsection we mention several variants and extensions of timed automata that have been proposed in the literature.

Timed automata as defined in this chapter are the so-called *diagonal-free* timed automata since only constraints of the form $x \bowtie k$ are used. Timed automata with diagonal constraints (of the form $x - y \bowtie k$) were also originally defined in the seminal paper [13]. They can be analyzed using

a slight refinement of the region automaton, but with no extra complexity. Furthermore, diagonal constraints can be removed from the model, at the expense of a (possibly exponential) blowup in the number of locations of the model [40].

Another useful extension of timed automata is obtained by allowing edges to set clocks to arbitrary positive integers ($x := k$) instead of only 0, or even to synchronize clock values ($x := y$). In [53] it is shown that any such *updatable* timed automaton can be converted to a usual one, hence this class is no more expressive than timed automata. If one also considers other updates however, like $x := x + 1$ or $x :> k$ (which non-deterministically sets x to some value larger than k), the situation is much more complex [53] and decidability of reachability is no longer preserved.

One can also extend the timed-automata formalism by allowing richer clock constraints, such as, e.g., $x + y \leq 5$ or $2x - 3y > 1$. Most such attempted extensions however lead to undecidability of the reachability problem, see for instance [41].

One can extend timed automata with *urgency* requirements [46]. For instance, some locations might be labelled as urgent, which indicates that no time can be spent in this location, it has to be left immediately when entered: an urgent location ℓ can easily be converted into a usual one by introducing an extra clock x which is reset in any edge to ℓ and has invariant $x = 0$ in ℓ, hence location-urgency does not add expressiveness to the class of timed automata. Some synchronization could be also labelled as urgent: in that case, the corresponding action should be done as soon as it is enabled. In our modeling of Example 1 the synchronization "acquire!/acquire?" could be made urgent since it is natural that a process acquires the resource as soon as it is available.

Another extension of timed automata we should mention is the *stopwatch* automata of [100]. Here timed automata are extended by allowing clocks to be stopped during a delay. Even though reachability is also undecidable for this extension and it has been shown to have the same expressive power as hybrid automata [72], stopwatch automata have found some applications, e.g., in scheduling [3, 141] and permit efficient over-approximate analysis [72]. Further extensions of the dynamics of timed automata lead to *rectangular* automata [100] and eventually to general *hybrid* automata [12, 98].

Finally, another interesting direction in which timed automata have been extended consists in adding parameters. Parameters can be used in lieu of numerical constants in the timed automaton, with the aim of deciding the existence of (and computing) values for the parameters for which a given property holds true. The use of parameters simplifies the modeling phase, but unfortunately the existence of valid parameters turns out to be undecidable in general [19]. Several decidable classes have been identified, including one-clock parametric timed automata [19, 129] and L/U automata, where each parameter can be used either in lower-bound constraints or in upper-bound constraints [106].

4 (Bi)simulation Checking

4.1 (Bi)simulations for Timed Automata

As detailed in Sect. 2, the operational semantics of timed automata is given in terms of timed transition systems, which in fact can be viewed as standard labelled transition systems, with labels (d, a) comprising a delay and a letter. Hence any behavioral equivalence and preorder defined on labelled transition systems may be interpreted over timed automata. In particular the classical notions of simulation and bisimulation [130, 138] give rise to the following notion of timed (bi)simulation:

Definition 4. Let $A = (L, \ell_0, C, \Sigma, I, E)$ be a timed automaton. A relation $R \subseteq L \times \mathbb{R}^C_{\geq 0} \times L \times \mathbb{R}^C_{\geq 0}$ is a *timed simulation* provided that for all $(\ell_1, v_1)\ R\ (\ell_2, v_2)$, for all $(\ell_1, v_1) \xrightarrow{d,a} (\ell_1', v_1')$ with $d \in \mathbb{R}_{\geq 0}$ and $a \in \Sigma$, there exists some (ℓ_2', v_2') such that $(\ell_1', v_1')\ R\ (\ell_2', v_2')$ and $(\ell_2, v_2) \xrightarrow{d,a} (\ell_2', v_2')$.

A *timed bisimulation* is a timed simulation which is also symmetric, and two states $(\ell_1, v_1), (\ell_2, v_2) \in [\![A]\!]$ are said to be *timed bisimilar*, written $(\ell_1, v_1) \sim (\ell_2, v_2)$, if there exists a timed bisimulation R for which $(\ell_1, v_1)\ R\ (\ell_2, v_2)$.

Note that $\sim$ is itself a timed bisimulation on A (indeed the greatest such), which is easily shown to be an equivalence relation and hence transitive, reflexive, and symmetric. Also—as usual—timed bisimilarity may be lifted to an equivalence between two timed automata A and B by relating their initial states.

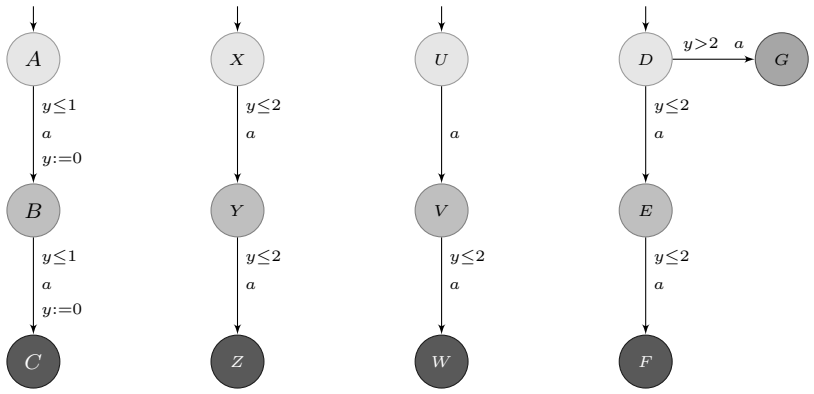


Fig. 6 Four timed automata A, X, U and D

Consider the four automata A, X, U and D in Fig. 6 (identifying the automata with the names of their initial locations). Here (U, v) and (D, v)

are timed bisimilar as any transition $(U,v) \xrightarrow{d,a} (V,v')$ may be matched by either $(D,v) \xrightarrow{a} (G,v')$ or $(D,v) \xrightarrow{a} (E,v')$ depending on whether $v(y) > 2$ or not after delay d. In fact, it may easily be seen that U and D are the only locations of Fig. 6 that are timed bisimilar (when coupled with the same valuation of y). E.g., A and X are not timed bisimilar since the transition $(X,0) \xrightarrow{1.5,a} (Y,1.5)$ cannot be matched by $(A,0)$ by a transition with *exactly* the same duration. Instead A and X are related by the weaker notion of *time-abstracted* bisimulation, which does not require equality of the delays (see Definition 2). It may be seen that A and X are both time-abstracted simulated by U and D but *not* time-abstracted bisimilar to U and D. Also, U and D are time-abstracted bisimilar, which follows from the following easy fact:

Theorem 2. *Any two automata being timed bisimilar are also time-abstracted bisimilar.*

4.2 Checking (Bi)simulations

As we now explain, timed and time-abstracted (bi)similarity are decidable for timed automata.

Theorem 3. *Time-abstracted similarity and bisimilarity are decidable for timed automata.*

For proving this result, one only needs to see that time-abstracted (bi)simulation in the timed automaton is the same as ordinary (bi)simulation in the associated region automaton; indeed, any state in $[\![A]\!]$ is untimed bisimilar to its image in $[\![A]\!]_{\simeq}$. The result follows by finiteness of the region automaton.

For timed bisimilarity, decidability—as we shall see in Sect. 9—is obtained by playing a game on a product construction, yielding an exponential-time algorithm for checking timed bisimilarity.

Theorem 4 ([73]). *Timed similarity and bisimilarity are decidable for timed automata.*

5 Language-Theoretic Properties

5.1 Language of a Timed Automaton

This section introduces the notion of (timed) *language* associated with timed automata, and focusses on basic decision problems such as language emptiness and inclusion, as well as standard Boolean operations on languages.

Properties of languages associated with various computational models are a classical object of study in computer science; moreover, many model-checking, refinement, and verification problems can often be stated in terms of languages, notably by translating them into language emptiness or language inclusion problems.

In this section we consider timed automata augmented with sets of *accepting locations*. Given a timed automaton $A = (L, \ell_0, C, \Sigma, I, E, F)$, where $F \subseteq L$ is the set of accepting locations, a finite run

$$\rho = (\ell_0, v_0) \xrightarrow{d_1, a_1} (\ell_1, v_1) \xrightarrow{d_2, a_2} \dots \xrightarrow{d_n, a_n} (\ell_n, v_n)$$

of A is *accepting* if $\ell_n \in F$. The language $\mathcal{L}(A)$ of A consists of all finite timed words over alphabet Σ^* generated by accepting runs of A.

The language of infinite words accepted by a timed automaton is defined analogously; the relevant acceptance condition is that the underlying infinite run visits locations in F infinitely often. We write $\mathcal{L}_\omega(A)$ to denote the set of infinite timed words accepted by A.

5.2 Timed Automata with ε-Transitions

Silent transitions are transitions of the form $(q, g, \varepsilon, r, q')$, where ε is the empty word. In other terms, they are transitions carrying no letter. Silent transitions offer a convenient way of modeling, e.g., internal actions. In the setting of finite-state automata, it is well known that such transitions can be removed, by merging them with the possible subsequent actions.

The question whether the above result extends to the timed setting was settled in [42], with a negative answer: to see this, simply consider the automaton in Fig. 7; its language $\mathcal{L}(A)$ contains precisely those timed words in which all timestamps are even integer numbers. Towards a contradiction, assume that there exists a timed automaton B, without ε-transitions, such that $\mathcal{L}(A) = \mathcal{L}(B)$; write m for the maximal integer constant appearing in the timing constraints of B. Then the one-letter word $(\sigma, 2m)$ is accepted by B, since it is accepted by A. Since B has no silent transition, it must have a σ-transition from an initial state to an accepting one. The guard on this transition can only involve constants less than or equal to m, so that B must accept (σ, k) for all $k > m$, which is a contradiction.

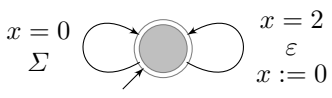


Fig. 7 A timed automaton with ε-transitions

Theorem 5 ([42]). *Silent transitions strictly increase the expressive power of timed automata.*

It can be proved that in the case when ε-transitions do not reset any clock, they do not add expressiveness. Finally, let us mention that the question whether a timed automaton with silent transitions is equivalent (i.e., accepts the same language) to some timed automaton without such transitions is undecidable [56].

In the sequel, we consider timed automata without ε-transitions.

5.3 Clock Constraints as Acceptance Conditions

Clock constraints can be used to enable or disable certain conditions along the runs of a timed automaton. As such, they can be used to define acceptance conditions, when added on top of a finite-state automaton.

In this setting, we consider the *untimed language* of timed automata: given a timed automaton A, its untimed language (of infinite words) is the set $\mathcal{L}_u$ containing exactly those words $(a_i)_{i\in\mathbb{N}}$ for which there is a *diverging* real-valued sequence $(d_i)_{i\in\mathbb{N}}$ such that the timed word $(a_i, d_i)_{i\in\mathbb{N}} \in \mathcal{L}_\omega(A)$. Notice that thanks to the time-abstracted bisimulation between a timed automaton and its region automaton, the untimed language of a timed automaton is easily seen to be ω-regular.

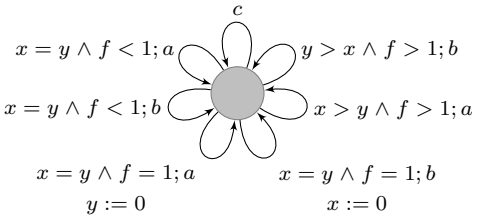


Fig. 8 Automaton accepting words with finitely many a's or finitely many b's

Conversely, any ω-regular language is the untimed language of a timed automaton: as an example, consider the language of infinite words over $\{a, b, c\}$ that have finitely many a's or finitely many b's. The untimed language of the automaton depicted in Fig. 8 precisely corresponds to that language: indeed, the a- and b-transitions on the left can only be taken finitely many times, since we require time divergence. Hence one of the a- and the b-transitions at the bottom has to be taken. But after this time, only the corresponding transition on the right is allowed (together with the c-transition, which is always allowed). This construction can be generalized, so that:

Theorem 6 ([101]). *Given an ω-regular language L, there exists a (one-location) timed automaton A such that $\mathcal{L}_u(A) = L$.*

The number of clocks and locations can be shown to define strict hierarchies of (untimed) ω-regular languages.

More generally, given a finite-state automaton A and an ω-regular language L, one can equip A with clocks and clock constraints in such a way that the untimed language of the resulting timed automaton is the intersection of the language of A with L [101].

5.4 Intersection, Union, and Complement

A (finite or infinite) language is said to be *timed regular* if it is accepted by some timed automaton. Timed regular languages (both finite and infinite) are effectively closed under intersection and union. They are however not closed under complement. We reproduce in Fig. 9 an example (taken from [13]) of a timed automaton A, equipped with a single clock, that cannot be complemented: there does not exist a timed automaton A' such that $\mathcal{L}_\omega(A')$ is the set of all timed words not accepted by A. The complement of $\mathcal{L}_\omega(A)$ contains

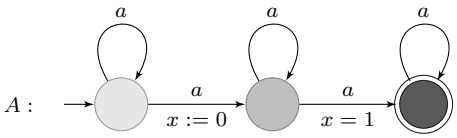


Fig. 9 A non-complementable timed automaton

all timed traces in which no pair of a's is separated by exactly one time unit. Intuitively, since there is no bound on the number of a's that can occur in any unit-duration time interval, any timed automaton capturing the complement of $\mathcal{L}_\omega(A)$ would require an unbounded number of clocks to keep track of the times of all the a's within the past one time unit. A formal proof that A cannot be complemented is given in [105].

Under some restrictions, timed automata can be made determinizable (hence also complementable). Most notably, *event-clock automata* [16] enjoy this property. In such timed automata, each letter a of the alphabet is associated with two clocks x_a and y_a (and any clock is associated with some letter that way): clock x_a (called the *event-recording clock* of a) is used to measure the delay elapsed since the last reset of event a (and is initially set to some special value $+\infty$), while y_a (the *event-predicting clock* of a) is used to constrain the delay until the next occurrence of a. One can easily show that event-clock automata can be represented as classical timed automata, though several clocks might be needed to encode each event-recording clock. Figure 10 displays an example of an event-clock automaton accepting those

timed words containing two *consecutive* a's separated by exactly one time unit.

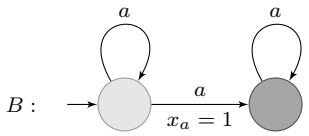


Fig. 10 An event-clock automaton

It must be remarked that, at any time during a run of an event-clock automaton on some timed word w, the valuation of the clocks does not depend on the run, but only on w. As a consequence, the classical subset construction for determinising finite-state automata can be adapted to handle event-clock automata, which thus form an (effectively) determinizable and complementable class of timed automata.

5.5 Language Emptiness, Inclusion

It follows immediately from Theorem 1 that the language-emptiness problem for timed automata is PSPACE-complete [13]. Unfortunately the language-universality, language-inclusion and language-equivalence problems for timed automata are all undecidable. By contrast, recall from Theorem 4 that the related branching-time counterparts to language inclusion and equivalence, namely similarity and bisimilarity, are both decidable on timed automata.

Theorem 7 ([13]). *The language-inclusion problem for timed automata is undecidable, both over finite and infinite words.*

The proof of Theorem 7 is by reduction from the Halting Problem for Turing machines. This reduction involves encoding the valid halting computations of a given Turing machine M as a timed language whose complement is recognized by a timed automaton A_M which can be effectively computed from M. Intuitively, discrete computation steps of M are simulated over unit-duration time intervals, with timed events used to encode the tape's contents. The integrity of the tape in a valid encoding of a computation is maintained by requiring that any given timed event be preceded and followed at a distance of exactly one time unit by the same timed event, and vice-versa (unless the corresponding character is to be modified by M in the computation step). Figure 11 illustrates this encoding. Note that the density of time enables one to accommodate arbitrarily large tape contents. The key idea is that while no timed automaton can in general accurately capture the encodings of valid computations of a Turing machine, A_M can be engineered to recognise

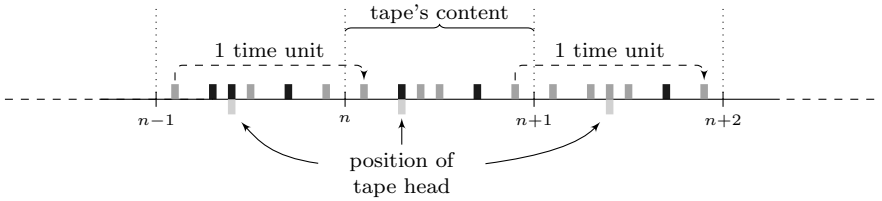


Fig. 11 Encoding computations of Turing machines as timed words

precisely all the *invalid* computations of M; indeed, a computation is invalid if it fails one of finitely many rules, the most interesting of which is to adequately preserve the tape's contents. The latter is easily detected, either by a timed automaton witnessing a timed event with no predecessor one time unit earlier (corresponding to an *insertion* error on the tape), or conversely by a timed automaton witnessing a timed event with no successor one time unit later (corresponding to a *deletion* error on the tape). Other rule failures can likewise be detected by small timed automata. Automaton A_M is obtained as the disjunction of those finitely many timed automata. The upshot is that M fails to have a valid halting computation iff A_M accepts every single timed trace. This shows that universality, and *a fortiori* language inclusion, are indeed undecidable for timed automata.

Theorem 7 places a serious limitation on the algorithmic analysis of timed automata since many verification questions naturally reduce to checking language inclusion. In spite of this hindrance there has been a great deal of research on various aspects of timed language inclusion, including [16,102,134] among many others. Here we describe several approaches, involving syntactic and semantic restrictions on timed automata, to obtaining positive decidability results for language inclusion.

Let us first notice that the classical approach to deciding language inclusion is by testing emptiness of the intersection of the first language with the complement of the second one. Thus whether $\mathcal{L}_\omega(A) \subseteq \mathcal{L}_\omega(B)$ is decidable as soon as B can be complemented. For instance:

Theorem 8 ([15]). *Given timed automata A and B, the language-inclusion problem $\mathcal{L}_\omega(A) \subseteq \mathcal{L}_\omega(B)$ is decidable if B is an event-clock automaton.*

Using more elaborate techniques (based on *well-quasi-orderings* and Higman's Lemma), we can prove:

Theorem 9 ([124, 134, 135]). *Given timed automata A and B, the finite-word language-inclusion problem $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ is decidable and non-primitive recursive provided B has at most one clock.*

In the case of infinite words, the language-inclusion problem $\mathcal{L}_\omega(A) \subseteq \mathcal{L}_\omega(B)$ is undecidable even when B has only one clock. The proof of undecidability is by reduction from the boundedness problem for lossy channel machines [4].

A natural semantic restriction on timed automata to recover decidability of language inclusion involves adopting a discrete-time model. Given a timed language $\mathcal{L}$, let $\mathbb{Z}(\mathcal{L})$ denote the set of timed words $(a_1, t_1) \dots (a_p, t_p)$ such that each timestamp t_i lies in $\mathbb{Z}$. Given timed automata A and B, the discrete-time language-inclusion problem is to decide whether $\mathbb{Z}(\mathcal{L}(A)) \subseteq \mathbb{Z}(\mathcal{L}(B))$. This problem is EXPSPACE-complete: the exponential blow-up over the complexity of the language-inclusion problem for classical non-deterministic finite automata arises from the succinct binary representation of clock values in timed automata. Hardness in EXPSPACE is proven in [28].

Using a technique called *digitization* [102] the discrete behaviors of timed automata can be used to infer conclusions about their dense-time behavior. For example:

Theorem 10 ([102]). *Let A be a closed timed automaton (i.e., having only non-strict inequalities as clock constraints) and B an open timed automaton (i.e., having only strict inequalities). Then $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ if and only if $\mathbb{Z}(\mathcal{L}(A)) \subseteq \mathbb{Z}(\mathcal{L}(B))$.*

To apply Theorem 10 one can imagine over-approximating a real-time model by a closed timed automaton and under-approximating a specification by an open timed automaton.

Rather than restricting the precision of the semantics we can instead consider a time-bounded semantics in which we consider only finite timed words of total duration at most N. Note that due to the density of time there is no bound on the number of events that can be performed in a fixed time period. In this case, for the whole class of timed automata, we have:

Theorem 11 ([133]). *Over bounded time (i.e., considering only finite timed words of total duration at most N, for some fixed time bound N), the language-inclusion problem is* 2-EXPSPACE*-complete.*

Theorem 11 was proven as a corollary of the decidability of satisfiability of monadic second-order logic over structures of the form $(I, <, +1)$, with I a bounded interval of reals and $+1$ denoting the plus-one relation: $+1(x, y)$ iff $y = x + 1$.

Notwithstanding the positive decidability results Theorem 9 and Theorem 11, neither one-clock timed automata nor automata over bounded time are closed under complement. (The counterexample in Sect. 5.4 can still be used in both cases.) To remedy this deficiency, the strictly more powerful model of *alternating timed automata* has been introduced [124, 135]. Alternating timed automata are a common generalization of timed automata and alternating finite automata; they are closed under all Boolean operations, but language inclusion remains decidable for alternating timed automata with one clock or over bounded time. Unlike in the untimed setting, alternating timed automata are strictly more expressive than purely non-deterministic timed automata. This extra expressiveness is crucially utilised in [135] where it is shown how to translate formulas of Metric Temporal Logic (see Sect. 6) into equivalent one-clock alternating timed automata.

6 Timed Temporal Logics

The whole theory of temporal-logic model checking has been extended to the setting of timed automata, in order to express and check richer properties beyond emptiness/reachability. We present the most significant results below.

6.1 Linear-Time Temporal Logics

The most natural way of extending LTL (see Chap. 2temp) with quantitative requirements is by decorating modalities with timing constraints. We present the resulting logic, called *Metric Temporal Logic* (MTL), below. Another extension consists in using clocks in formulas, with a way of resetting them when some property is fulfilled and checking their values at a later moment. The resulting logic is called *Timed Propositional Temporal Logic* (TPTL). Due to lack of space, we don't detail the latter logic, and rather refer to [17, 18, 52] for more details about TPTL.

Given a set P of atomic propositions, the formulas of MTL are built from P using Boolean connectives and time-constrained versions of the *until* operator $\mathbf{U}$ as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathbf{U}_I\, \varphi\,,$$

where $I \subseteq (0,\infty)$ is an interval of reals with endpoints in $\mathbb{N} \cup \{\infty\}$. We sometimes abbreviate $\mathbf{U}_{(0,\infty)}$ to $\mathbf{U}$, calling this the *unconstrained* until operator.

Further connectives can be defined following standard conventions. In addition to propositions $\top$ (true) and $\bot$ (false) and disjunction $\vee$, we have the *constrained eventually* operator $\Diamond_I \varphi \equiv \top \,\mathbf{U}_I\, \varphi$, and the *constrained always* operator $\Box_I\, \varphi \equiv \neg\Diamond_I \neg\varphi$.

Sometimes MTL is presented with past connectives (e.g., constrained versions of the "since" connective from LTL) as well as future connectives [17]. However we do not consider past connectives in this chapter.

Next we describe two commonly adopted semantics for MTL.

Continuous Semantics.

Given a set of propositions P, a *signal* is a function $f\colon \mathbb{R}_{\geq 0} \to 2^P$ mapping $t \in \mathbb{R}_{\geq 0}$ to the set $f(t)$ of propositions holding at time t. We say that f has *finite variability* if its set of discontinuities has no accumulation points (in other words, on any finite interval the value of f can only change a finite number of times). In this chapter, we require that all signals be finitely variable. Given an MTL formula φ over the set of propositional variables P, the satisfaction relation $f \models \varphi$ is defined inductively, with the classical rules for atomic

propositions and Boolean operators, and with the following rule for the "until" modality, where f^t denotes the signal $f^t(s) = f(t+s)$:

$$f \models \varphi_1 \,\mathbf{U}_I\, \varphi_2 \;\text{ iff }\; \text{for some } t \in I,\ f^t \models \varphi_2 \text{ and } f^u \models \varphi_1 \text{ for all } u \in (0,t).$$

Pointwise Semantics.

In the *pointwise semantics* MTL formulas are interpreted over timed words. Given a (finite or infinite) timed word $w = (a_1, t_1), \ldots, (a_n, t_n)$ over alphabet 2^P and an MTL formula φ, the satisfaction relation $w, i \models \varphi$ (read "w satisfies φ at position i") is defined inductively, with the classical rules for Boolean operators, and with the following rule for the "until" modality:

$$w, i \models \varphi_1 \,\mathbf{U}_I\, \varphi_2 \;\text{ iff }\; \begin{array}{l}\text{there exists } j \text{ such that } i < j < |w|,\ w, j \models \varphi_2,\\ \qquad t_j - t_i \in I, \text{ and } w, k \models \varphi_1 \text{ for all } k \text{ with } i < k < j.\end{array}$$

The pointwise semantics is less natural if one thinks of temporal logics as encoding fragments of monadic logic over the reals. On the other hand it seems more suitable when considering MTL formulas as specifications on timed automata. In this vein, when adopting the pointwise semantics it is natural to think of atomic propositions in MTL as referring to events (corresponding to location changes) rather than to locations themselves.

Consider our example of Fig. 2. Using LTL, we can express the property that Process P_i will try to get the resource infinitely many times, by writing $\Box\Diamond\,\texttt{start!}$. With MTL, we can be more precise and write $\Box\Diamond_{\leq 4}\,\texttt{start!}$, stating that whatever the current state, within four time units Process P_1 will start trying to acquire the resource. MTL can also be used to express bounded-time response properties, such as $\Box\,(\texttt{start!} \Rightarrow \Diamond_{\leq 10}\,\texttt{acquire!})$.

6.2 Verification of Linear-Time Temporal Logics

Model checking timed automata can be carried out under either pointwise or continuous semantics. For the latter, it is necessary to alter our definitions to associate a language of *signals* with a timed automaton rather than a language of timed words. In turn, this requires a notion of timed automata in which locations are labelled by atomic propositions. A full development of this semantics can be found, e.g., in [14].

Theorem 12 ([13]). *Model checking and satisfiability for LTL, over both the pointwise and continuous semantics, are* PSPACE*-complete.*

The PSPACE upper bound in Theorem 12 can be established in the same manner as in the untimed case, by translating the negated formula to a Büchi automaton, and performing an on-the-fly reachability check on the product of this automaton with the region graph of the model.

Theorem 13 ([135–137]). *Model checking and satisfiability for MTL in the pointwise semantics over finite words are decidable but non-primitive recursive. Over infinite words, both problems are undecidable.*

As explained in Sect. 5.5, the decidability results are essentially obtained by translating MTL formulas into one-clock alternating timed automata, and rephrasing the model-checking or satisfiability problems as instances of language emptiness in one-clock alternating timed automata.

The undecidability result proceeds by reduction from the recurrent reachability problem for channel machines with insertion errors: the infinite runs of such a machine can be encoded as timed words, which in turn are easily characterized by an MTL formula [136].

Theorem 14 ([14]). *Model checking and satisfiability for MTL in the continuous semantics (over both finite and infinite signals) are undecidable.*

The extra expressiveness of the continuous semantics enables a more direct proof of undecidability than in Theorem 13. In this case, one can directly encode the computations of a Turing machine as timed signals, which again can be captured by an MTL formula, following a scheme similar to that of Theorem 7.

A key ingredient of the undecidability proof of Theorem 14, as well as the non-primitive recursive complexity in Theorem 13, is the ability of MTL to express *punctuality*, i.e., the requirement that two events be separated by an exact duration. A fragment of MTL that syntactically disallows punctuality, known as *Metric Interval Temporal Logic* (MITL), was proposed by Alur *et al.* in [14]. In MITL, one requires that all instances of the interval I appearing in uses of the constrained until operator $\mathbf{U}_I$ be non-singular. The main result of [14] is as follows:

Theorem 15 ([14]). *For both the pointwise and continuous semantics, model checking and satisfiability for MITL are* EXPSPACE*-complete, over both finite and infinite behaviors.*

This theorem was obtained by translating MITL formulas into equivalent timed automata of potentially exponential size. In contrast, it is easy to write an MTL formula that has no equivalent timed automaton: for example, the formula $\neg\Diamond\,(a \wedge \Diamond_{=1}\, a)$ captures the complement of the language of automaton A in Fig. 9.

Another decidable fragment of MTL can be obtained by adapting the idea of *event clocks* to temporal logics [140]: here, timing constraints can only refer to the *next* (or *previous*) occurrence of an event. Hence ECTL (standing for *Event-Clock Temporal Logic*) extends LTL with $\triangleright_I \varphi$ and $\triangleleft_I \varphi$. For instance, that there are two *consecutive* a's separated by one time unit is written in ECTL as $\Diamond\,(a \wedge \triangleright_{=1} a)$.

Theorem 16 ([140]). *Satisfiability and model checking are* PSPACE*-complete for ECTL, in either the pointwise or continuous semantics.*

Not surprisingly, deciding ECTL is achieved by a translation to event-clock automata (see Sect. 5.4). However, event-clock automata are not powerful enough to precisely capture ECTL, and require the use of *timed Hintikka sequences*. Intuitively, given a formula φ in ECTL, a timed Hintikka sequence is a timed word on sets of subformulas of φ, required to satisfy local consistency conditions. Compare to a timed word, a timed Hintikka sequence contains more information about the truth value of the subformulas of φ, which will help the event-clock automaton decide whether the underlying timed word is to be accepted. We refer to [140] for more details, and to [104] for an extension of event-clock automata that encompasses ECTL.

Rather than imposing syntactical restrictions, an alternative approach to recovering decidability is to consider a *time-bounded* semantics, i.e., in which either timed words or signals are observed over a fixed, bounded time interval:

Theorem 17 ([133]). *Model checking and satisfiability for MTL over bounded time, in either the pointwise or continuous semantics, are* EXPSPACE*-complete.*

The main technique used in the proof of Theorem 17 is an exponential transformation from MTL formulas, given a fixed time bound, into equisatisfiable LTL formulas.

6.3 Branching-Time Temporal Logics

Given a set P of atomic propositions, TCTL* formulas are state-formulas obtained as formulas φ_s from the following grammar:

$$\varphi_s ::= p \mid \neg\varphi_s \mid \varphi_s \wedge \varphi_s \mid \mathbf{E}\varphi_p \mid \mathbf{A}\varphi_p$$
$$\varphi_p ::= \varphi_s \mid \neg\varphi_p \mid \varphi_p \wedge \varphi_p \mid \varphi_p \,\mathbf{U}_I\, \varphi_p$$

Compared to CTL*, TCTL* has a *time-constrained* until, requiring as for MTL that the right-hand side formula should be fulfilled within that time. The same shorthands as for MTL can be defined, such as $\Diamond_I\varphi$ or $\Box_I\varphi$.

As for the linear-time temporal logics, the semantics of TCTL* comes in (at least) two flavours: continuous and pointwise. The continuous semantics is defined on *dense trees*, which naturally extend classical discrete trees to the continuous setting [7, 89] and represent the set of signals of timed automata; this semantics extends the continuous-time semantics of MTL with path quantifiers.

The pointwise semantics is defined over discrete (but infinite-branching) trees, which can be used to represent the timed words generated by timed automata. This corresponds to evaluating TCTL* formulas over the operational semantics of timed automata, as defined in Sect. 2.

Finally, as for the untimed case, the fragment of TCTL* where each temporal modality is under the immediate scope of a path quantifier is of particular interest, and will be called TCTL.

Before turning to the algorithmic part, let us show how TCTL can be used to express desirable properties of the timed system modeled in Fig. 2. Mutual exclusion (in a setting with two processes P_1 and P_2 and one resource R_1) is expressed as $\neg\, \mathbf{E}\Diamond\,(\text{working}_1 \wedge \text{working}_2)$: two processes will never be working (i.e., using the resource) at the same time. We can also express timing requirements, such as the fact that Process P_i will never be working continuously for more than one time unit: $\mathbf{A}\Box\,(\,\mathbf{A}\Diamond_{\leq 1}\,\neg\,\text{working}_i)$.

6.4 Verification of Branching-Time Temporal Logics

Since TCTL* embeds MTL, there is no hope that its model checking and satisfiability will be decidable. On the lower side, CTL model checking is clearly decidable over timed automata: CTL is invariant under bisimulation, so that any property to be checked on a timed automaton can equivalently be checked on its corresponding finite-state region automaton.

Concerning TCTL, model checking can be shown to be decidable. Actually, this can easily be shown on the explicit-clock version of TCTL (using formula clocks), which strictly subsumes TCTL. For this logic, the important property is that any two region-equivalent states satisfy the same formulas: this can be proven by induction on the structure of the formula. Consider for instance the formula $\zeta = \mathbf{E}\varphi\,\mathbf{U}_I\,\psi$, assuming that φ and ψ are compatible with region (i.e., if they hold true in some state, then they also hold true in any region-equivalent state). Given a timed automaton A, consider the automaton A_t obtained by adding an extra clock t to A, which does not modify its behavior. Then if $(\ell, v) \models \zeta$ in A, then $(\ell, v_t) \models \mathbf{E}\varphi\,\mathbf{U}\,(\psi \wedge t \in I)$ where v_t extends v by mapping clock t to zero and "$t \in I$" (which is not a TCTL formula but whose meaning is rather clear) is also compatible with regions. In the end, the set of states in A_t where formula $\mathbf{E}\varphi\,\mathbf{U}\,(\psi \wedge t \in I)$ is a union of regions, so that it is also the case for the set of states of A where ζ holds.

Using this result, TCTL model checking can be achieved by labeling states of the region automaton with the subformulas they satisfy. This applies for both semantics, with slight differences. It should be noticed that this extends to the explicit-clock version of TCTL, and even to the fragment of explicit-clock TCTL* where formula-clocks are only reset at the level of path quantifiers [66].

While labeling the region automaton requires exponential space, the algorithm can be implemented in a space-efficient manner so as to only use polynomial space. Reachability being already PSPACE-hard, we get the following theorem:

Theorem 18 ([9]). *TCTL model checking is* PSPACE*-complete.*

TCTL (as well as CTL) suffers from not being able to express useful properties, in particular fairness (see Chap. 2temp on temporal logics). One way to solve this problem is by decorating path quantifiers with fairness requirements [7, 152]. One can then apply classical algorithms for CTL with fairness [75] or adapt fixpoint characterizations. Another approach is to consider TCTL defined with formula clocks as sketched above, and to have it include CTL^*. The resulting logic is very expressive while still enjoying a PSPACE model-checking algorithm [66].

These positive results about model checking do not extend to satisfiability:

Theorem 19 ([9]). *TCTL satisfiability is undecidable.*

The proof follows the same ideas as for undecidability of MTL satisfiability, by associating universal path-quantifiers with each temporal modality. As for the linear-time case, it suffices to ban equality constraints to recover decidability [117, 118]. This can be proved by lifting tree-automata techniques to the timed setting.

7 Symbolic Algorithms, Data Structures, Tools

7.1 Zones and Operations

As shown in the previous sections, the regions introduced in Sect. 3 provide a finite and elegant abstraction of the infinite state space of timed automata, enabling us to prove decidability of a wide range of problems, including (timed and untimed) bisimilarity, untimed language equivalence and language emptiness, as well as TCTL model checking.

Unfortunately, the number of states obtained from the region partitioning is extremely large. Indeed, it is exponential in the number of clocks as well as in (the binary representation of) the maximal constants of the timed automaton [13]. Efforts have been made to develop more efficient representations of the state space [34, 39, 103, 121], using the notion of *zones* introduced below as a coarser and more compact representation of the state space.

For a finite set C of clocks, a subset $Z \subseteq \mathbb{R}^C_{\geq 0}$ is called a *zone* if there exists $\varphi \in \Phi_d(C)$ for which $Z = [\![\varphi]\!]_C$. For reachability analysis, we need the following operations on zones: for a zone $Z \subseteq \mathbb{R}^C_{\geq 0}$ and $r \subseteq C$, let us denote

- the *delay* of Z by $Z^{\uparrow} = \{v + d \mid v \in Z, d \in \mathbb{R}_{\geq 0}\}$ and
- the *reset* of Z under r by $Z[r] = \{v[r] \mid v \in Z\}$.

Lemma 1 ([103, 157]). *Let Z, Z' be zones over C and $r \subseteq C$. Then $Z^{\uparrow}$, $Z[r]$, and $Z \cap Z'$ are also zones over C.*

Definition 5. The *zone automaton* associated with a timed automaton $A = (L, \ell_0, C, \Sigma, I, E)$ is the transition system $[\![A]\!]_Z = (S, s_0, \Sigma \cup \{\delta\}, T)$ given as follows:

$$S = \big\{(\ell, Z) \;\big|\; \ell \in L, Z \subseteq \mathbb{R}^C_{\geq 0} \text{ is a zone}\big\} \qquad s_0 = \big(\ell_0, [\![v_0]\!]\big)$$
$$T = \big\{(\ell, Z) \overset{\delta}{\rightsquigarrow} \big(\ell, Z^{\uparrow} \cap [\![I(\ell)]\!]_C\big)\big\}$$
$$\cup \big\{(\ell, Z) \overset{a}{\rightsquigarrow} \big(\ell', (Z \cap [\![\varphi]\!]_C)[r] \cap [\![I(\ell')]\!]_C\big) \;\big|\; \ell \xrightarrow{\varphi, a, r} \ell' \in E\big\}$$

Analogously to Prop. 1, we have:

Proposition 2 ([157]). *A location ℓ in a timed automaton $A = (L, \ell_0, F, C, \Sigma, I, E)$ is reachable if and only if there is a zone $Z \subseteq \mathbb{R}^C_{\geq 0}$ for which (ℓ, Z) is reachable in $[\![A]\!]_Z$.*

A priori, however, the zone automaton defined above is infinite, hence another, finite, abstraction is needed. This is provided by *normalization* using region equivalence $\cong_M$: for a maximal constant M, the *normalization* of a zone $Z \subseteq \mathbb{R}^C_{\geq 0}$ is the set $\{v : C \to \mathbb{R}_{\geq 0} \mid \exists v' \in Z : v \cong_M v'\}$. The normalization of a zone is not in general a zone, hence in practice other normalization operators are used (see Sect. 7.3).

The normalized zone automaton is defined analogously to the zone automaton defined above, and in case the timed automaton to be verified does not contain diagonal clock constraints of the form $x - y \bowtie k$, Prop. 2 also holds for the normalized zone automaton. Hence we can obtain a reachability algorithm by applying any search strategy (depth-first, breadth-first, or another) on the normalized zone automaton.

For timed automata that contain diagonal clock constraints $x - y \bowtie k$, however, it can be shown [38, 48] that normalization as defined above does *not* give rise to a sound and complete characterization of forward reachability. Instead, one can apply a refined normalization which depends on the difference constraints used in the timed automaton, see [38].

7.2 Symbolic Datastructures

A zone, given by a conjunction of elementary clock constraints, may be represented using a directed weighted graph, where the nodes correspond to the clocks in C together with an extra "zero" clock x_0, and an edge $x_i \xrightarrow{k} x_j$ corresponds to a constraint $x_i - x_j \leq k$ (if there is more than one upper bound on $x_i - x_j$, k is the minimum of all these constraints' right-hand sides). The extra clock x_0 is fixed at value 0, so that a constraint $x_i \leq k$ can be represented as $x_i - x_0 \leq k$. Lower bounds on $x_i - x_j$ are represented as (possibly negative) upper bounds on $x_j - x_i$, and strict bounds $x_i - x_j < k$ are represented by adding a flag to the corresponding edge.

The weighted graph in turn may be represented by its adjacency matrix, which in this context is known as a *difference-bound matrix* or DBM. The above technique was introduced in [87] (the main ideas were already present in [43]). Figure 12 gives an illustration of an extended clock constraint together with its representation as a difference-bound matrix. Note that the clock constraint contains superfluous information.

$$Z = \begin{cases} x_1 \leq 3 \\ x_1 - x_2 \leq 10 \\ x_1 - x_2 \geq 4 \\ x_1 - x_3 \leq 2 \\ x_3 - x_2 \leq 2 \\ x_3 \geq -5 \end{cases}$$

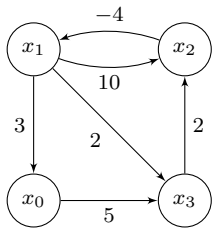


Fig. 12 Graph representation of extended clock constraint

On zones represented using DBMs, efficient (in time cubic in the number of clocks in C) algorithms are available for computing delays, resets and intersections. For reachability checking, other necessary operations inclusion checking whether $Z \subseteq Z'$, and emptiness checking, $Z = \emptyset$; these can also be computed efficiently using DBMs.

For these computations, a canonical representation obtained as the *shortest-path closure* of the DBM graph is used. Another useful canonical form is the *shortest-path reduction* described in [120]. Whereas the shortest-path closure form gives all derived constraints, the shortest-path reduced form provides a memory-efficient representation, containing only a minimal set of constraints. Figure 13 shows the two canonical forms of the DBM of Fig. 12. Given the shortest-path closure of the DBM graph, the shortest-path reduced form is obtained by partitioning the set of clocks according to zero cycles in the DBM graph; in the DBM of Fig. 13, $\{x_1, x_2, x_3\}$ constitutes one such class, and $\{x_0\}$ is another one. The reduced form is now obtained by maintaining a minimal set of constraints for each class, essentially a simple cycle of the clocks of the class, and only keeping constraints between representatives of different classes, here x_1 and x_0.

To combat state-space explosion in zone graphs, several optimizations are used. One such approach is to detect whether a state (ℓ, Z) reached through the algorithm is contained in another state (ℓ, Z') which has already been explored. In this case, exploration of (ℓ, Z) will be unnecessary.

Another such optimization is to work with *unions of zones*. If states (ℓ, Z) and (ℓ, Z') are found to reachable during the analysis, then we know that altogether the state $(\ell, Z \cup Z')$ is reachable. Unfortunately, unions of zones are not generally themselves zones, so cannot be efficiently represented using DBMs.

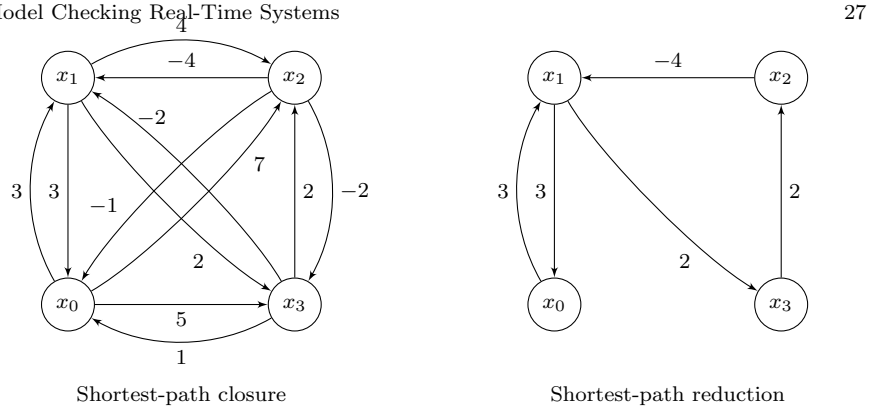


Fig. 13 Canonical representations

For representing unions of zones, or *federations* as they are called in this context, a data structure inspired by decision diagrams called *clock difference diagrams* or CDDs is used [121]. However, no efficient algorithms are known to compute delays or resets of federations using CDDs, so in practice reachability analysis using CDDs is done by extracting the zones from the CDD and performing the operations on them one by one.

Other promising data structures in this context are *numerical decision diagrams* [24], *difference decision diagrams* [132], *clock-restriction diagrams* [154], *max-plus polyhedra* [5], *constraint matrix diagrams* [88], and *time-darts* [111]; generally, the design of efficient data structures for symbolic exploration of timed automata is a field of active research.

7.3 Practical Efficiency

Symbolic, zone-based exploration of the (reachable) state-space of timed automata using the DBM data structure is key to their analysis. However, a number of additional algorithmic techniques have been developed for gaining efficiency in practice. In the following we indicate a number of these.

As described in Sect. 7.1, normalization of zones with respect to the maximum constant M appearing in the given timed automata ensures finiteness of the zone graph and hence termination of algorithms searching the zone graph. Here, a practical problem is that the normalization of a zone is in general not a zone itself, but rather a finite union of such. However, given a representation of the zone as a *shortest-path-closed* DBM, a syntactic *extrapolation* operation that removes bounds that are larger than the maximum constant may be easily performed: any upper bound constraint of the form $x_i - x_j < k$ where $k > M$ is removed, and any lower bound of the form $x_i - x_j > k$ where $k < -M$ is replaced by $x_i - x_j > -M$. Clearly, under extrapolation, only a finite

number of DBMs will be encountered, ensuring termination. Furthermore, the correctness is based on the fact that extrapolation of a zone (based on its DBM representation) is included in its normalization, as shown in [29, 48].

Coarser, yet complete, notions of extrapolation have been obtained by performing the operation with respect to *clock- and location-dependent* maximum constants [29] and further differentiating the maximum constant used in upper or lower bound comparisons [30, 31]. In fact, this last extrapolation yields performance comparable to that of the overapproximate *convex hull* abstraction [81].

Several of the algorithmic problems presented in this chapter—e.g., reachability, model checking, and equivalence checking, as well as notions of optimal reachability and controllability, which will be described in later sections—have a fixed-point characterization. Though easy to implement, this leads to backwards iterative algorithms, requiring one to consider and classify states which are possibly not even reachable. Thus, for most problems, so-called *on-the-fly* algorithms have been devised, where the satisfaction of the given property by the initial state is attempted to be concluded in as *local* a fashion as possible, only exploring the state space when needed. For the analysis of timed-automata-based models, such on-the-fly algorithms have been proposed for instance for reachability [120, 157], liveness checking [147, 151], model checking with respect to TCTL [47], time-abstract bisimulation checking [149], and controller synthesis for timed game automata [69].

Despite the above efforts in applying aggressive (yet complete) abstractions, the analysis of timed-automata-based models suffers from the state-space explosion problem. Thus, complementary techniques have been proposed and implemented for reducing space consumption at the expense of time performance [35, 120]. Also, various AI-inspired techniques have been developed for *efficient guidance* of the symbolic exploration of timed automata towards specified error states [114–116]. Similarly, the technique of *symmetry reduction* has been developed and implemented for networks of timed automata with several symmetric components, potentially yielding an exponential gain in performance [96]. Moreover, so-called *time-convexity* analysis provides significant performance improvement [153].

Finally, several attempts have been made to extend the technique of *partial-order reduction* to networks of timed automata. In contrast to the proved effect in the finite-state setting, early attempts [37, 131] did not show any improvement in performance compared with regular symbolic exploration. In fact, being a strong synchronizer in a timed-automata network, time reduces the number of independent transitions, which are key for partial-order reduction to have effect. In later work [36,125,126], it was found that the union of all zones reached by different interleavings of the same set of transitions is convex, providing the basis of substantial improvement.

Also, bounded model-checking techniques have been developed for timed automata using *difference logics*, though with a limited performance improvement [76, 77].

7.4 Tools and Applications

Timed automata and their extensions have been applied to the modeling, analysis and optimization of numerous real-time applications. In this section we give a few examples, not aiming at being exhaustive but rather to illustrate the wide range of application domains.

A variety of mature tools are available which provide important computer-aided support for applications. Well-known tools include UppAal [122], Kronos [158], RED [155] and HyTech [99]. A larger number of other tools related to the analysis of timed automata have emerged over the years including Else [159], Rabbit [44], Verics [84], and TAME [22] as well as tools for analyzing other timed formalisms based on translation to timed automata including Times [21] (task automata), Moby [145] (PLC programs), SART [45] (Safety Critical Java), ART [93] (task graphs), Romeo [91] and TAPALL [68] (Time and Timed-arc Petri Nets), and VeSTA [112] (component integration checking).

The timed-automata formalism is now routinely applied to the modeling and analysis of real-time control programs, including a wide class of Programmable Logic Controller (PLC) control programs [85,127] and timing analysis and code generation of vehicle control software [150]. The timed-automata approach has also demonstrated its viability in the timing analysis of certain classes of asynchronous circuits [60].

Similarly, numerous real-time communication protocols have been analyzed using timed automata technology, often with inconsistencies being revealed: e.g., using real-time model checking, the cause of a ten-year-old bug in the IR-link protocol used by Bang & Olufsen was identified and corrected [95]. Most recently, real-time model checking has been applied to the clock synchronization algorithm currently used in a wireless sensor network that has been developed by the Dutch company CHESS [144]. Here it is shown that in certain cases a static, fully synchronized network may eventually become unsynchronized if the current algorithm is used, even in a setting with infinitesimal clock drifts.

During the last few years, timed automata modeling of multitasking applications running under real-time operating systems has received substantial research effort. Here the goals are multiple: to obtain less-pessimistic worst-case response time analysis compared with classical methods for single-processor systems [156]; to relax the constraints of period task arrival times of classical scheduling theory to task arrival patterns that can be described using timed automata [90]; to allow for schedulability analysis of tasks in terms of concurrent objects executing on multiprocessor or distributed platforms (e.g., MPSoC) [61, 80, 109].

Just as symbolic reachability checking of finite-state models has led to very efficient planning and scheduling algorithms, reachability checking for (priced) timed automata has demonstrated competitive and complementary performance with respect to classical approaches such as MIPL on optimal scheduling problems involving real-time constraints, e.g., job-shop and

task-graph scheduling [1, 32] and aircraft landing problems [119]. In fact a translation of the variant PDDL3 of PDDL (Planning Domain Definition Language) into priced timed automata has been made [86] allowing optimal planning questions to be answered by cost-optimal reachability checking. Industrial applications include planning a wafer scanner from the semiconductor industry [97] and computation of optimal paper paths for printers [108].

Most recently, computation of winning strategies for timed games has been applied to controller synthesis for embedded systems, including synthesis of most general non-preemptive online schedulers for real-time systems with sporadic tasks [6], synthesis of climate control for pig shed provided by the company Skov [110], and automatic synthesis of robust and near-optimal controllers for industrial hydraulic pumps [71].

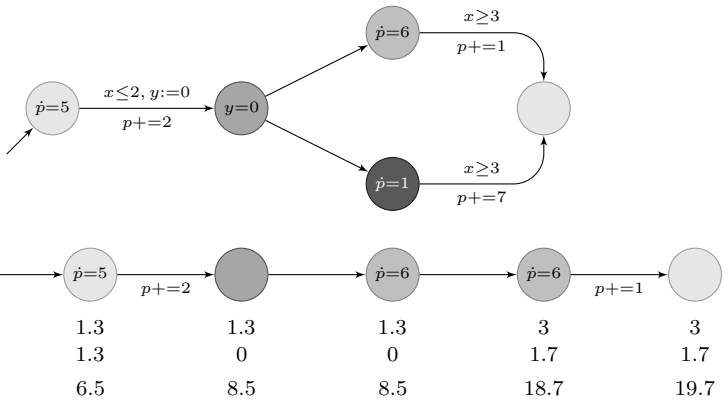
8 Weighted Timed Automata

Time is not the only quantity one may want to measure when checking an embedded system: one may need to keep track of the battery charge or of the level of oil in a tank. Hybrid automata [98, 100] extend timed automata with extra variables that can help measure such quantities. Unfortunately, reachability is undecidable for these models, even with two-slope hybrid variables. Weighted timed automata [20, 33] is an intermediary model, extending timed automata with hybrid *observer* variables: these variables cannot appear in the guards of the automaton, but they can be used, for example, for optimization purposes. The special case where the observer is a stopwatch (computing the accumulated delay in some locations) was already introduced and solved in [11].

Formally, a *weighted timed automaton* is a pair (A, w) where A is a timed automaton and w labels the locations and edges of A with an integer (or a vector of integers for automata with multiple observer variables). For a transition t, $w(t)$ is the value by which the value of the observer variable is increased, while for a location ℓ, $w(\ell)$ is the *rate* by which the variable increases w.r.t. time (in other words, the observer variable p follows the differential equation $dp/dt = w(\ell)$).

The semantics of a weighted timed automaton (A, w) is that of the underlying timed automaton A. Each run of A is decorated with the value of the observer variable. Figure 14 shows an example of a weighted timed automaton[3], and a run of this automaton. This run reaches the rightmost location within 3 time units, and with a total weight of 19.7.

[3] Notice that the labeling in the second location is a clock invariant (enforcing that no time will elapse in that location). The rate of p is not given in that location as no time will elapse anyway.



x	0	1.3	1.3	1.3	3	3
y	0	1.3	0	0	1.7	1.7
p	0	6.5	8.5	8.5	18.7	19.7

Fig. 14 Example of a weighted timed automaton

8.1 Cost-Optimal Schedules

Natural questions on this family of models include optimal reachability of a given location, or optimal mean-cost of infinite runs. Formally, the associated decision problems respectively ask whether the target location can be reached with total weight less than a given threshold, and whether there is an infinite run along which the average weight is less than the given bound.

These problems turn out to be decidable. The main technique used in the algorithms is a refinement of the region equivalence of Sect. 3 called *corner-point equivalence* [50]. Intuitively, for these two optimization problems, optimal schedules will amount to staying *as long as possible* in interesting locations. Corner-point regions extend classical regions with an extra *corner* of this region, i.e., an integer valuation that belongs to the closure of the region. A corner-point (r, c) represents a clock valuation v that is *close* to the corner c but lies within the region r. The *corner-point automaton* is the weighted automaton $\mathcal{C}_{\simeq M} = (S, s_0, \Sigma, T)$ where $S \subseteq L \times \mathcal{R} \times \{0, ..., M\}^C$ is the set of states (writing $\mathcal{R}$ for the set of regions), with three kinds of transitions:

- action transitions: there is a transition from (ℓ, R, c) to (ℓ', R', c') if there is a transition $t = (\ell, \varphi, a, r, \ell')$ in A such that $R \subseteq [\![\varphi]\!]_C$, with $R' = R[r]$ and $c' = c[r]$. The weight of this transition is $w(t)$.
- ε-delay transitions: these are transitions from (ℓ, R, c) to (ℓ, R', c), where R' is the immediate time-successor of R sharing corner c. Such a transition corresponds to a very small delay, and its corresponding weight is set to zero.
- 1-delay transitions: these are transitions from (ℓ, R, c) to (ℓ, R, c'), where $c' = c + 1$. This corresponds to spending (almost) one time unit in re-

gion (ℓ, R). Notice that such a transition is only available if c and $c+1$ are corners of R. The weight of this transition is $w(\ell)$.

Figure 15 displays two sequences of delay transitions in the corner-point automaton; while both sequences depart from the same region and visit the same sequence of locations, the accumulated weight evolves very differently along the two sequences—which explains why we have to refine regions.

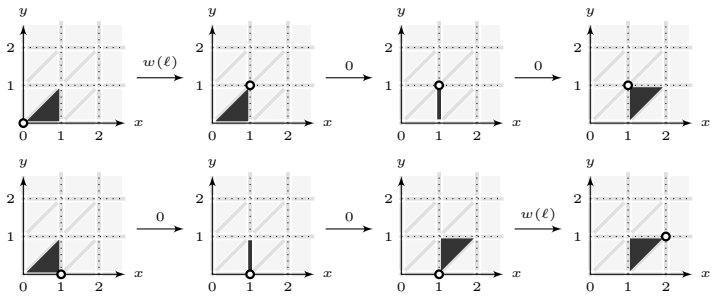


Fig. 15 Two different runs in the corner-point abstraction

The corner-point automaton enjoys the following properties: if there is a run from some location ℓ to some location ℓ' with total weight m in a given weighted timed automaton, then there is a run from $(\ell, \mathbf{0}, \mathbf{0})$ to some (ℓ', R, c) with total weight at most m in the corresponding corner-point automaton; in other words, running through corner-points is always better when trying to optimize the value of the total weight. Conversely, for any positive ε, if there is a run from $(\ell, \mathbf{0}, \mathbf{0})$ to some (ℓ', R, c) with weight m in the corner-point automaton, then there is a run in the original weighted timed automaton from ℓ (with initial valuation $\mathbf{0}$) to ℓ', with total weight at most $m + \varepsilon$.

This statement can be extended in various ways, so as to handle optimization of the ratio between two variables along finite or infinite runs. In the end:

Theorem 20 ([50]). *The optimal-reachability and the optimal-ratio problems are* PSPACE*-complete.*

Other related problems have been considered in the literature, such as conditional optimal reachability on multi-weighted timed automata [123]. The aim in this setting is to minimize the value of one variable under some conditions on the other variables. We refer to [123], where the problem is shown to be decidable.

8.2 Weighted Temporal Logics

Unfortunately, the encouraging results above do not extend to richer properties that could be expressed in weighted extensions of classical temporal logics[4]. While this is not surprising for linear-time temporal logics (as these logics are already mostly undecidable in the timed setting), this also holds for weighted extensions of CTL, be it with modalities decorated with weight constraints (writing $\mathbf{E}\Diamond_{\leq 10} T$ to express that T can be reached within total cost less than 10), or with explicit constraints as atomic formulas (writing $\mathbf{E}\Diamond (T \wedge c \leq 10)$ to express the same property) [49,63]. Undecidability can be proved by encoding the halting problem for a two-counter machine, where each counter is encoded by a clock of the timed automaton. The central trick in the reduction is the ability to multiply the value of a clock by some constant (while preserving the value of the other clocks). This is achieved by the automaton depicted in Fig. 16, in which we enforce the condition that location T must be reachable from S with total cost exactly 1: indeed, the total cost accumulated from S to T is precisely $1 + 2x_0 - y_0$, where x_0 and y_0 are the values of clocks x and y in S. This provides us with a way of doubling the value of clock x, by letting clock y play the role of x afterwards. Using this module, it is easy to build a complete reduction involving four clocks, which can be further improved to only three clocks.

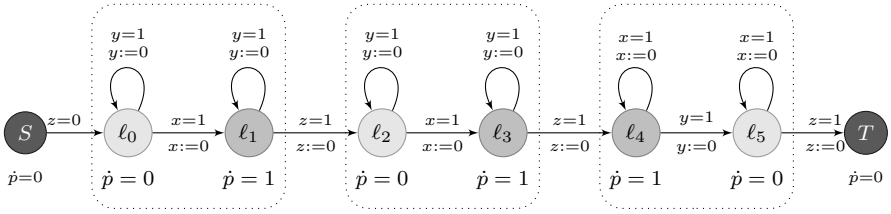


Fig. 16 Module for testing whether $y = 2x$

Theorem 21 ([49,63]). *WCTL model checking is undecidable (on weighted timed automata with at least three clocks).*

Notice that the standard restriction to non-punctual constraints in the logic does not help, as the above reduction can be carried out using only inequality constraints. One way of recovering decidability is to restrict to *one-clock* weighted timed automata. These automata enjoy several special properties which allow us to prove that refining regions to a small granularity (in $O(C^{-h(\varphi)})$ where C is the maximal rate in the automaton and $h(\varphi)$ is the temporal height of the formula being checked) provides a correct finite-state abstraction on which φ can be checked. It follows:

[4] Even if we restrict to nonnegative weights, which is what we assume in this subsection on temporal logics.

Theorem 22 ([57]). *WCTL model checking is* PSPACE*-complete on one-clock weighted timed automata.*

8.3 Energy Constraints

Recently, weighted timed automata have been pushed one step closer to hybrid automata, with the introduction of *energy constraints* [55,74]. These constraints aim at modeling, for example, autonomous robots, which often must take care of their battery charge level, and ensure that they never run out of energy. This is modeled with weighted timed automata, with the constraint that the accumulated value of the variable must never drop below 0 (or any lower bound). The same problem can of course be considered with an additional upper-bound constraint. Notice that this is a departure from the motto that the cost variable is an observer. Figure 17 displays an example of a weighted timed automaton, together with the evolution of the variable along one particular run. This corresponds to a *feasible* (prefix of a) run, as the variable remains between the lower bound L and upper bound U.

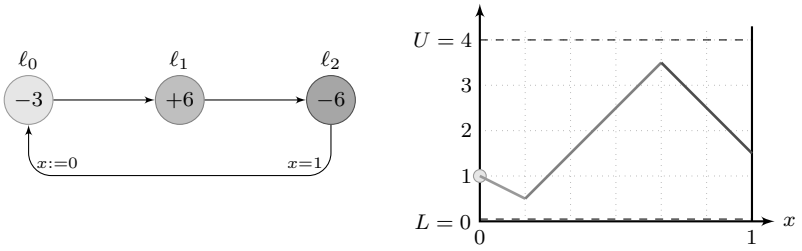


Fig. 17 A weighted timed automaton under energy constraints

Only a few results have been obtained so far. Let us begin with the untimed case [55]: for lower-bound constraints, the problem still amounts to optimizing the accumulated cost, with the extra *energy constraint*. In that setting, the Bellman–Ford algorithm can be used to compute the maximal accumulated cost that can be achieved from the initial state, with the extra energy constraint. This provides a polynomial-time algorithm for solving reachability under lower-bound constraints. In case we also have an upper-bound constraint, the problem can be solved in exponential time by augmenting the state space with the explicit value of the variable.

In the timed setting, the only known positive results concern one-clock weighted timed automata under lower-bound constraints [54]. The

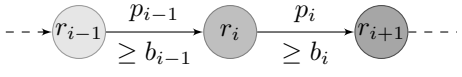


Fig. 18 A *linear* weighted automaton

central technique is the computation of an optimal schedule through a finite *linear* automaton (i.e., visiting all its locations with a fixed, linear order), such as the one depicted in Fig. 18. Notice that along such runs, we allow *lower-bound constraints* (written $\geq b_i$) on each transition. Along such a path, one can prove that the optimal policy is to spend no time in a location r_i if

- either $r_{i-1} > r_i$ (in which case it is more profitable to spend time in location r_{i-1});
- or $r_{i+1} \geq r_i$ and $b_{i-1} + p_{i-1} \geq b_i$ (in which case it is possible to directly jump to the more profitable location r_{i+1}).

If any of these conditions is fulfilled, location r_i can be dropped, and replaced by a transition from r_{i-1} to r_{i+1} with weight $p_{i-1} + p_{i+1}$ and cost-constraint $\geq \max(b_{i-1}, b_i - p_{i-1})$. This provides us with a linear automaton along which the rates are increasing. The optimal policy along such a path can be proved to be to exit a location as soon as the cost constraint $\geq b_i$ is fulfilled. This gives a way of computing the optimal achievable energy level at the end of the path as a function of the initial credit. This extends to one-clock automata by composing such functions. In the end:

Theorem 23 ([54]). *Optimal reachability is decidable in one-clock weighted timed automata under lower-bound constraints.*

Unfortunately, this algorithm does not extend to n-clock automata: indeed, one can easily come up with a small module to increase or decrease the value of the cost variable by the value of a clock (see Fig. 16), thus providing a way of checking linear constraints between several clocks. As a consequence:

Theorem 24 ([58]). *Optimal reachability is undecidable in four-clock weighted timed automata under lower-bound constraints.*

Weighted timed automata under energy constraints are a very recent and active topic with many open problems. Several directions are currently being explored, such as the extension to *exponential variables*, where the variable follows the differential equation $dp/dt = w(\ell) \cdot p$, or the inclusion of imprecisions in clock values or variable growth.

9 Timed Games

Games provide a nice framework for modeling and reasoning about the interactions between various agents (a reactive system and its environment, several components, etc.). We refer to Chap. 25gamesynth for details about games and their use for synthesizing correct models.

We consider two-player timed games, in which transitions are partitioned into controllable and uncontrollable (i.e., under the control of an environment).

The problem is then to synthesize a strategy telling *when* to take *which* (enabled) controllable transitions in order that a given objective is guaranteed regardless of the behavior of the environment.

Definition 6. A *timed game* is a tuple $G = (L, \ell_0, C, \Sigma_c, \Sigma_u, I, E)$ with $\Sigma_c \cap \Sigma_u = \emptyset$ for which the tuple $A_G = (L, \ell_0, C, \Sigma = \Sigma_c \cup \Sigma_u, I, E)$ is a timed automaton. We require this automaton to be deterministic (so that from any state, an action in Σ corresponds to a unique transition). Edges with actions in Σ_c are said to be *controllable*, those with actions in Σ_u are *uncontrollable*.

We shall again assume a set F of accepting locations to be given for the rest of this section. A *strategy* in a timed game G provides instructions as to which controllable edge to take, or whether to wait, in a given state. Hence it is a mapping σ from finite runs of the underlying timed automaton A_G to $\Sigma_c \cup \{\delta\}$, where $\delta \notin \Sigma$, such that for any run $\rho = (\ell_0, v_0) \to \cdots \to (\ell_k, v_k)$,

- if $\sigma(\rho) = \delta$, then $(\ell, v) \xrightarrow{d} (\ell, v + d)$ is a transition in $[\![A_G]\!]$ for some $d > 0$, and
- if $\sigma(\rho) = a$, then $(\ell, v) \xrightarrow{a} (\ell', v')$ is a transition in $[\![A_G]\!]$.

An *outcome* of a strategy is any run which adheres to its instructions. Such an outcome is said to be *maximal* if it stops in an accepting location, or if no controllable actions are available at its end. An underlying assumption is that uncontrollable actions cannot be forced, hence a maximal outcome which does not end in a final location may "get stuck" in a non-final location. The aim of reachability games is to find strategies all of whose maximal outcomes end in an accepting location; the aim of safety games is to find strategies all of whose (not necessarily maximal) outcomes avoid accepting locations:

Definition 7. A strategy is said to be *winning for the reachability game* if any of its maximal outcomes is an accepting run. It is said to be *winning for the safety game* if none of its outcomes are accepting.

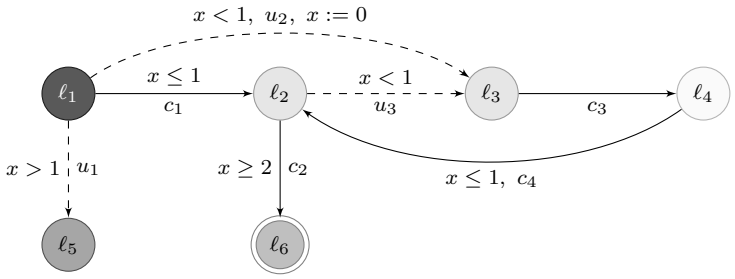


Fig. 19 A timed game with one clock. Controllable edges (with actions from Σ_c) are solid, uncontrollable edges (with actions from Σ_u) are dashed

Example 5. Figure 19 provides a simple example of a timed game. Here, $\Sigma_c = \{c_1, c_2, c_3, c_4\}$ and $\Sigma_2 = \{u_1, u_2, u_3\}$, and the controllable edges are drawn with solid lines, the uncontrollable ones with dashed lines. The following memoryless strategy is winning for the reachability game:

$$\sigma(\ell_1, v) = \begin{cases} \delta & \text{if } v(x) \neq 1 \\ c_1 & \text{if } v(x) = 1 \end{cases} \qquad \sigma(\ell_2, v) = \begin{cases} \delta & \text{if } v(x) < 2 \\ c_2 & \text{if } v(x) \geq 2 \end{cases}$$

$$\sigma(\ell_3, v) = \begin{cases} \delta & \text{if } v(x) < 1 \\ c_3 & \text{if } v(x) \geq 1 \end{cases} \qquad \sigma(\ell_4, v) = \begin{cases} \delta & \text{if } v(x) \neq 1 \\ c_4 & \text{if } v(x) = 1 \end{cases}$$

Theorem 25 ([25, 26, 65, 128]). *The (time-optimal) reachability and safety games are decidable for timed games. They are* EXPTIME*-complete.*

A key ingredient in the proof of the above theorem is the fact that for reachability as well as safety games, it is sufficient to consider *memoryless* strategies, which only observe the last configuration of a run. This is not the case for other, more subtle, control objectives (e.g., counting properties modulo some N) as well as for the synthesis of winning strategies under *partial observability*. The other ingredient is the region abstraction: if there is a winning strategy, then there is one that only depends on the current region. This provides an exponential-time algorithm, which can be proved to be optimal.

A problem with the above approach is that the safety game can be won by preventing time from diverging. In order to rule out such behaviors, a solution was proposed in [82]; it uses a more symmetric presentation of games, in which both players have a strategy which proposes at the same time the amount of time this player wants to delay, and the transition she wants to take afterwards. At each step, the player with the shortest delay is chosen and her choice is performed. With this definition, if time converges along an outcome, then the player(s) who have applied their choices infinitely many times must have proposed converging sequences of delays. By adding a fairness requirement to the winning condition, we can declare this kind of behavior losing. Deciding the existence of winning strategies for reachability and safety objectives remains EXPTIME-complete in this context.

The field of timed games is a very active research area. From the region-based decidability results, efficient on-the-fly algorithms have been developed [69, 148] and implemented in UPPAAL. In [70] these algorithms have been extended to timed games under partial observability. Research has also been conducted towards the synthesis of *optimal* winning strategies for reachability games on *weighted timed games*. In [8, 51] computability of optimal strategies is shown under a certain condition of *strong cost non-Zenoness*, requiring that the total weight diverges at a given minimum rate per time. Later undecidability results [49, 64] show that for weighted timed games with three or more clocks, this condition (or a similar one) is necessary. It is proved

in [59] that optimal reachability strategies are computable for one-clock weighted timed games, though there is an unsettled (large) gap between the known lower bound complexity P and an upper bound of 3-EXPTIME, which was recently lowered to EXPTIME [94, 142].

We conclude this section by illustrating how timed games can be used to decide timed bisimilarity of two states of a timed automaton. This provides a simple proof of Theorem 4, which we explain on a small example: consider the states of Fig. 20. That two states (p, v) and (q, w) (where v and w are two

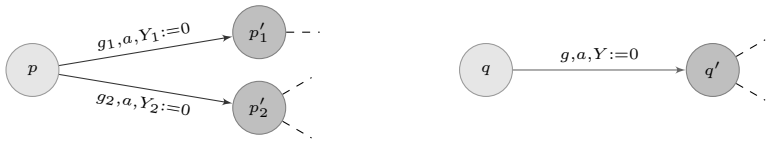


Fig. 20 Two states of a timed automaton

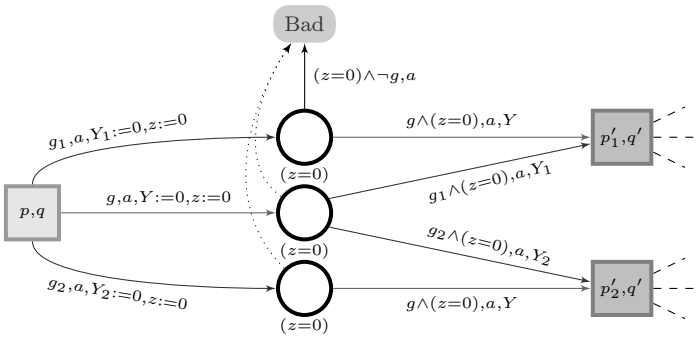


Fig. 21 Timed bisimilarity as a timed game

valuations of the same set of clocks $\mathcal{C}$) are timed bisimilar means that any transition from either state can be mimicked from the other one, ending up in states that are again bisimilar. We can see this as a game on the product of two copies of the automaton (see Fig. 21): from the joint state $((p, q), (v, w))$, the first player chooses to apply one transition from one of the states (p, v) and (q, w), and the second player has to respond (immediately) with an appropriate move from the other state. The second player has a strategy to always avoid the Bad state if, and only if, the starting states are timed bisimilar. This provides an exponential-time algorithm for checking timed bisimilarity [10].

10 Ongoing and Future Challenges

In this chapter, we have surveyed timed automata and the theoretical developments that have led to their being widely accepted as a model for modeling and reasoning about real-time systems. Many developments are still ongoing: we briefly list here some important topics which we think are among the important avenues to be explored during the coming years:

- Robustness (in the timed setting) and implementability [83, 92, 139, 143] address the problem of reconciling the semantics of timed automata (with real-valued clocks) with the models they represent (which usually run at a finite frequency).
- Statistical model checking consists in checking several runs of the model against a given property, and compute statistics to get an estimate of the correctness of the model. The approach has recently been studied and implemented in the setting of stochastic timed automata, where it provides interesting results, even for problems that are otherwise undecidable [79].
- Games on timed automata have received a lot of attention over the last ten years, as they are a convenient formalism for the automated synthesis of real-time systems. Recent extensions to non-zero-sum games [62, 113], where the players have their own objectives, open a rich and promising area of research for synthesizing complex systems.

Acknowledgement

We thank the reviewers for their numerous comments, remarks and additional references, which greatly helped us improve this chapter.

References

1. Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006.
2. Yasmina Abdeddaïm and Oded Maler. Job-shop scheduling using timed automata. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Intl. Conf. on Computer-Aided Verification (CAV)*, volume 2102 of *Lecture Notes in Computer Science*, pages 478–492. Springer, 2001.
3. Yasmina Abdeddaïm and Oded Maler. Preemptive job-shop scheduling using stopwatch automata. In Joost-Pieter Katoen and Perdita Stevens, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2280 of *Lecture Notes in Computer Science*, pages 113–126. Springer, 2002.
4. Parosh Aziz Abdulla, Johann Deneux, Joël Ouaknine, and James Worrell. Decidability and complexity results for timed automata via channel machines. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors,

International Colloquium on Automata, Languages and Programming (ICALP), volume 3580 of *Lecture Notes in Computer Science*, pages 1089–1101. Springer, 2005.
5. Xavier Allamigeon, Stéphane Gaubert, and Éric Goubault. Inferring min and max invariants using max-plus polyhedra. In María Alpuente and Germán Vidal, editors, *Intl. Symp. on Static Analysis (SAS)*, volume 5079 of *Lecture Notes in Computer Science*, pages 189–204. Springer, 2008.
6. Karine Altisen, Gregor Gößler, Amir Pnueli, Joseph Sifakis, Stavros Tripakis, and Sergio Yovine. A framework for scheduler synthesis. In *Symposium on Real-Time Systems (RTSS)*, pages 154–163. IEEE, 1999.
7. Rajeev Alur. *Techniques for Automatic Verification of Real-Time Systems*. PhD thesis, Stanford University, 1991.
8. Rajeev Alur, Mikhail Bernadsky, and Parthasarathy Madhusudan. Optimal reachability for weighted timed games. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3142 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.
9. Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
10. Rajeev Alur, Costas Courcoubetis, and Thomas A. Henzinger. The observational power of clocks. In Bengt Jonsson and Joachim Parrow, editors, *Intl. Conf. on Concurrency Theory (CONCUR)*, volume 836 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 1994.
11. Rajeev Alur, Costas Courcoubetis, and Thomas A. Henzinger. Computing accumulated delays in real time systems. *Formal Methods in System Design (FMSD)*, 11(2):137–155, 1997.
12. Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems (HSCC)*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer, 1993.
13. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
14. Rajeev Alur, Tómas Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.
15. Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. In David L. Dill, editor, *Intl. Conf. on Computer-Aided Verification (CAV)*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1994.
16. Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211(1-2):253–273, 1999.
17. Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.
18. Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–203, 1994.
19. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Symposium on the Theory of Computing (STOC)*, pages 592–601. ACM, 1993.
20. Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovani-Vincentelli, editors, *Int. Workshop on Hybrid Systems: Computation and Control (HSCC)*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2001.

21. Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times – a tool for modelling and implementation of embedded systems. In Joost-Pieter Katoen and Perdita Stevens, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2280 of *Lecture Notes in Computer Science*, pages 460–464. Springer, 2002.
22. Myla Archer, Constance L. Heitmeyer, and Elvinia Riccobene. Using TAME to prove invariants of automata models: Two case studies. In Mats Per Erik Heimdahl, editor, *Workshop on Formal Methods in Software Practice (FMSP)*, pages 25–36. ACM, 2000.
23. André Arnold and Maurice Nivat. The metric space of infinite trees. algebraic and topological properties. *Fundamenta Informaticae*, 3(4):445–476, 1980.
24. Eugene Asarin, Marius Bozga, Alain Kerbrat, Oded Maler, Amir Pnueli, and Anne Rasse. Data-structures for the verification of timed automata. In Oded Maler, editor, *International Workshop on Hybrid and Real-Time Systems (HART)*, volume 1201 of *Lecture Notes in Computer Science*, pages 346–360. Springer, 1997.
25. Eugene Asarin and Oded Maler. As soon as possible: Time optimal control for timed automata. In Frits Vaandrager and Jan H. van Schuppen, editors, *Int. Workshop on Hybrid Systems: Computation and Control (HSCC)*, volume 1569 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.
26. Eugene Asarin, Oded Maler, and Amir Pnueli. Symbolic controller synthesis for discrete and timed systems. In Panos Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II (HSCC)*, volume 999 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 1995.
27. Eugene Asarin, Oded Maler, and Amir Pnueli. On discretization of delays in timed automata and digital circuits. In Davide Sangiorgi and Robert de Simone, editors, *Intl. Conf. on Concurrency Theory (CONCUR)*, volume 1466 of *Lecture Notes in Computer Science*, pages 470–484. Springer, 1998.
28. Christel Baier, Nathalie Bertrand, Patricia Bouyer, and Thomas Brihaye. When are timed automata determinizable? In Susanne Albers, Marchetti-Spaccamela Alberto, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5557 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2009.
29. Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim Gulstrand Larsen. Static guard analysis in timed automata verification. In Hubert Garavel and John Hatcliff, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–270. Springer, 2003.
30. Gerd Behrmann, Patricia Bouyer, Kim Gulstrand Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. In Kurt Jensen and Andreas Podelski, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2988 of *Lecture Notes in Computer Science*, pages 312–326. Springer, 2004.
31. Gerd Behrmann, Patricia Bouyer, Kim Gulstrand Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *Intl. Journal on Software Tools for Technology Transfer (STTT)*, 8(3):204–215, 2006.
32. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Gulstrand Larsen, Paul Pettersson, and Judi Romijn. Efficient guiding towards cost-optimality in Uppaal. In Tiziana Margaria and Wang Yi, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2031 of *Lecture Notes in Computer Science*, pages 174–188. Springer, 2001.
33. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Gulstrand Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovani-Vincentelli, editors, *Int. Workshop on Hybrid Systems: Computation and Control*

(HSCC), volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

34. Gerd Behrmann, Kim Gulstrand Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Efficient timed reachability analysis using clock difference diagrams. In Nicolas Halbwachs and Doron A. Peled, editors, *Intl. Conf. on Computer-Aided Verification (CAV)*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353. Springer, 1999.
35. Gerd Behrmann, Kim Gulstrand Larsen, and Radek Pelánek. To store or not to store. In Warren A. Hunt, Jr and Fabio Somenzi, editors, *Intl. Conf. on Computer-Aided Verification (CAV)*, volume 2725 of *Lecture Notes in Computer Science*, pages 433–445. Springer, 2003.
36. Ramzi Ben Salah, Marius Bozga, and Oded Maler. On interleaving in timed automata. In Christel Baier and Holger Hermanns, editors, *Intl. Conf. on Concurrency Theory (CONCUR)*, volume 4137 of *Lecture Notes in Computer Science*, pages 465–476. Springer, 2006.
37. Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In Davide Sangiorgi and Robert de Simone, editors, *Intl. Conf. on Concurrency Theory (CONCUR)*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer, 1998.
38. Johan Bengtsson and Wang Yi. On clock difference constraints and termination in reachability analysis of timed automata. In Jin Song Dong and Jim Woodcock, editors, *International Conference on Formal Engineering Methods (ICFEM)*, volume 2885 of *Lecture Notes in Computer Science*, pages 491–503. Springer, 2003.
39. Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 2098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2004.
40. Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2-3):145–182, 1998.
41. Béatrice Bérard and Catherine Dufour. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1-2):1–7, 2000.
42. Béatrice Bérard, Paul Gastin, and Antoine Petit. Timed automata with non-observable actions: Expressive power and refinement. In Claude Puech and Rüdiger Reischuk, editors, *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1046 of *Lecture Notes in Computer Science*, pages 257–268. Springer, 1996.
43. Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In R. E. A. Mason, editor, *IFIP World Computer Congress (WCC)*, pages 41–46. North-Holland/IFIP, 1983.
44. Dirk Beyer, Claus Lewerentz, and Andreas Noack. Rabbit: A tool for BDD-based verification of real-time systems. In Warren A. Hunt, Jr and Fabio Somenzi, editors, *Intl. Conf. on Computer-Aided Verification (CAV)*, volume 2725 of *Lecture Notes in Computer Science*, pages 122–125. Springer, 2003.
45. Thomas Bøgholm, Henrik Kragh-Hansen, Petur Olsen, Bent Thomsen, and Kim Gulstrand Larsen. Model-based schedulability analysis of safety critical hard real-time Java programs. In Gregory Bollella and C. Douglas Locke, editors, *International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES)*, volume 343, pages 106–114. ACM, 2008.
46. Sébastien Bornot, Joseph Sifakis, and Stavros Tripakis. Modeling urgency in timed systems. In Willem-Paul de Roever, Hans Langmaack, and Amir Pnueli, editors, *International Symposium on Compositionality: The Significant Difference (COMPOS)*, volume 1536 of *Lecture Notes in Computer Science*, pages 103–129. Springer, 1998.
47. Ahmed Bouajjani, Stavros Tripakis, and Sergio Yovine. On-the-fly symbolic model checking for real-time systems. In *Symposium on Real-Time Systems (RTSS)*, pages 25–35. IEEE, 1997.

48. Patricia Bouyer. Untameable timed automata! In Helmut Alt and Michel Habib, editors, *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631. Springer, 2003.
49. Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5):188–194, 2006.
50. Patricia Bouyer, Ed Brinksma, and Kim Gulstrand Larsen. Staying alive as cheaply as possible. *Formal Methods in System Design (FMSD)*, 32(1):2–23, 2008.
51. Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim Gulstrand Larsen. Optimal strategies in priced timed game automata. In Kamal Lodaya and Meena Mahajan, editors, *Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, 2004.
52. Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. *Information and Computation*, 208(2):97–116, 2010.
53. Patricia Bouyer, Catherine Dufour, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.
54. Patricia Bouyer, Uli Fahrenberg, Kim Gulstrand Larsen, and Nicolas Markey. Timed automata with observers under energy constraints. In Karl Henrik Johansson and Wang Yi, editors, *Int. Workshop on Hybrid Systems: Computation and Control (HSCC)*, pages 61–70. ACM, 2010.
55. Patricia Bouyer, Uli Fahrenberg, Kim Gulstrand Larsen, Nicolas Markey, and Jiři Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008.
56. Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Undecidability results for timed automata with silent transitions. *Fundamenta Informaticae*, 92(1-2):1–25, 2009.
57. Patricia Bouyer, Kim Gulstrand Larsen, and Nicolas Markey. Model checking one-clock priced timed automata. *Logicical Methods in Computer Science*, 4(2), 2008.
58. Patricia Bouyer, Kim Gulstrand Larsen, and Nicolas Markey. Lower-bound constrained runs in weighted timed automata. In *Int. Conf. on Quantitative Evaluation of Systems (QEST)*, pages 128–137. IEEE, 2012.
59. Patricia Bouyer, Kim Gulstrand Larsen, Nicolas Markey, and Jacob Illum Rasmussen. Almost optimal strategies in one-clock priced timed automata. In S. Arun-Kumar and Naveen Garg, editors, *Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.
60. Marius Bozga, Hou Jianmin, Oded Maler, and Sergio Yovine. Verification of asynchronous circuits using timed automata. In Eugene Asarin, Oded Maler, and Sergio Yovine, editors, *Workshop on Theory and Practice of Timed Systems (TPTS)*, volume 65 of *Electronic Notes in Theoretical Computer Science*, pages 47–59. Elsevier Science, 2002.
61. Aske Wiid Brekling, Michael R. Hansen, and Jan Madsen. Models and formal verification of multiprocessor system-on-chips. *Journal of Logic and Algebraic Programming*, 77(1-2):1–19, 2008.
62. Romain Brenguier. *Nash Equilibria in Concurrent Games – Application to Timed Games*. PhD thesis, Lab. Spécification & Vérification, ENS Cachan, France, 2012.
63. Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. Model-checking for weighted timed automata. In Yassine Lakhnech and Sergio Yovine, editors, *Joint Int. Conf. on Formal Modelling and Analysis of Timed Systems (FORMATS) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, volume 3253 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2004.

64. Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In Paul Pettersson and Wang Yi, editors, *International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume 3829 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2005.
65. Thomas Brihaye, Thomas A. Henzinger, Vinayak S. Prabhu, and Jean-François Raskin. Minimum-time reachability in timed games. In Lars Arge, Christian Cachin, Tomasz Jurdziński, and Andrzej Tarlecki, editors, *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4596 of *Lecture Notes in Computer Science*, pages 825–837. Springer, 2007.
66. Thomas Brihaye, François Laroussinie, Nicolas Markey, and Ghassan Oreiby. Timed concurrent game structures. In Luís Caires and Vasco T. Vasconcelos, editors, *Intl. Conf. on Concurrency Theory (CONCUR)*, volume 4703 of *Lecture Notes in Computer Science*, pages 445–459. Springer, 2007.
67. Janusz A. Brzozowski and Carl-Johan H. Seger. Advances in asynchronous circuit theory part II: Bounded inertial delay models, MOS circuits, design techniques. *EATCS Bulletin*, 43:199–263, 1991.
68. Joakim Byg, Kenneth Yrke Jørgensen, and Jiři Srba. TAPAAL: Editor, simulator and verifier of timed-arc Petri nets. In Zhiming Liu and Anders P. Ravn, editors, *Intl. Symp. Automated Technology for Verification and Analysis (ATVA)*, volume 5799 of *Lecture Notes in Computer Science*, pages 84–89. Springer, 2009.
69. Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Gulstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *Intl. Conf. on Concurrency Theory (CONCUR)*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005.
70. Franck Cassez, Alexandre David, Kim Gulstrand Larsen, Didier Lime, and Jean-François Raskin. Timed control with observation based and stuttering invariant strategies. In Kedar Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *Intl. Symp. Automated Technology for Verification and Analysis (ATVA)*, volume 4762 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2007.
71. Franck Cassez, Jan J. Jensen, Kim Gulstrand Larsen, Jean-François Raskin, and Pierre-Alain Reynier. Automatic synthesis of robust and optimal controllers – an industrial case study. In Rupak Majumdar and Paulo Tabuada, editors, *Int. Workshop on Hybrid Systems: Computation and Control (HSCC)*, volume 5469 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2009.
72. Franck Cassez and Kim Gulstrand Larsen. The impressive power of stopwatches. In Catuscia Palamidessi, editor, *Intl. Conf. on Concurrency Theory (CONCUR)*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.
73. Kārlis Čerāns. Decidability of bisimulation equivalences for parallel timer processes. In Gregor von Bochmann and David K. Probst, editors, *Intl. Workshop on Computer-Aided Verification (CAV)*, volume 663 of *Lecture Notes in Computer Science*, pages 302–315. Springer, 1993.
74. Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *Int. Conf. on Embedded Software (EMSOFT)*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003.
75. Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
76. Scott Cotton, Eugene Asarin, Oded Maler, and Peter Niebert. Some progress in satisfiability checking for difference logic. In Yassine Lakhnech and Sergio Yovine, editors, *Joint Int. Conf. on Formal Modelling and Analysis of Timed Systems (FORMATS) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, volume 3253 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 2004.

77. Scott Cotton and Oded Maler. Fast and flexible difference constraint propagation for DPLL(T). In Armin Biere and Carla P. Gomes, editors, *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4121 of *Lecture Notes in Computer Science*, pages 170–183. Springer, 2006.
78. Alexandre David, Kim Gulstrand Larsen, Axel Legay, and Marius Mikučionis. Schedulability of Herschel-Planck revisited using statistical model checking. In Tiziana Margaria and Bernhard Steffen, editors, *International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, volume 7610 of *Lecture Notes in Computer Science*, pages 293–307. Springer, 2012.
79. Alexandre David, Kim Gulstrand Larsen, Axel Legay, Marius Mikučionis, and Zheng Wang. Time for statistical model checking of real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Intl. Conf. on Computer-Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 349–355. Springer, 2011.
80. Alexandre David, Kim Gulstrand Larsen, Jacob Illum Rasmussen, and Arne Skou. Model-based design for embedded systems. In Gabriela Nicolescu and Pieter J. Mosterman, editors, *Model-Based Design for Embedded Systems*, Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press, 2009.
81. Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In Bernhard Steffen, editor, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.
82. Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In Roberto Amadio and Denis Lugiez, editors, *Intl. Conf. on Concurrency Theory (CONCUR)*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003.
83. Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robust safety of timed automata. *Formal Methods in System Design (FMSD)*, 33(1-3):45–84, 2008.
84. Piotr Dembinski, Agata Janowska, Paweł Janowski, Wojciech Penczek, Agata Pólrola, Maciej Szreter, Bożna Woźna, and Andrzej Zbrzezny. Verics: A tool for verifying timed automata and estelle specifications. In Hubert Garavel and John Hatcliff, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2619 of *Lecture Notes in Computer Science*, pages 278–283. Springer, 2003.
85. Henning Dierks. PLC-automata: a new class of implementable real-time automata. *Theoretical Computer Science*, 253(1):61–93, 2001.
86. Henning Dierks. Finding optimal plans for domains with continuous effects with UPPAAL Cora. In *ICAPS Workshop on Verification and Validation of Model-Based Planning and Schedulin Systems (VV&PS)*, 2005.
87. David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In Joseph Sifakis, editor, *International Workshop on Automatic Verification Methods for Finite State Systems (AVMFSS)*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1990.
88. Rüdiger Ehlers, Daniel Fass, Michael Gerke, and Hans-Jörg Peter. Fully symbolic timed model checking using constraint matrix diagrams. In *Symposium on Real-Time Systems (RTSS)*, pages 360–371. IEEE Comp. Soc. Press, 2010.
89. Marco Faella, Salvatore La Torre, and Aniello Murano. Dense real-time games. In *Symp. on Logic in Computer Science (LICS)*, pages 167–176. IEEE, 2002.
90. Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theoretical Computer Science*, 354(2):301–317, 2006.
91. Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier H. Roux. Romeo: A tool for analyzing time Petri nets. In Kousha Etessami and Sriram Rajamani, editors, *Intl. Conf. on Computer-Aided Verification (CAV)*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423. Springer, 2005.

92. Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan. Robust timed automata. In Oded Maler, editor, *International Workshop on Hybrid and Real-Time Systems (HART)*, volume 1201 of *Lecture Notes in Computer Science*, pages 331–345. Springer, 1997.
93. Michael R. Hansen, Jan Madsen, and Aske Wiid Brekling. Semantics and verification of a language for modelling hardware architectures. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems, Essays in Honor of Dines Bjørner and Zhou Chaochen on the Occasion of Their 70th Birthdays*, volume 4700 of *Lecture Notes in Computer Science*, pages 300–319. Springer, 2007.
94. Thomas Dueholm Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. A faster algorithm for solving one-clock priced timed games. In Pedro R. D'Argenio and Hernán C. Melgratt, editors, *Intl. Conf. on Concurrency Theory (CONCUR)*, volume 8052 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2013.
95. Klaus Havelund, Arne Skou, Kim Gulstrand Larsen, and Kristian Lund. Formal modelling and analysis of an audio/video protocol: An industrial case study using Uppaal. In *Symposium on Real-Time Systems (RTSS)*, pages 2–13. IEEE, 1997.
96. Martijn Hendriks, Gerd Behrmann, Kim Gulstrand Larsen, Peter Niebert, and Frits Vaandrager. Adding symmetry reduction to Uppaal. In Kim Gulstrand Larsen and Peter Niebert, editors, *Int. Workshop on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume 2791 of *Lecture Notes in Computer Science*. Springer, 2003.
97. Martijn Hendriks, Barend van den Nieuwelaar, and Frits Vaandrager. Model checker aided design of a controller for a wafer scanner. *Intl. Journal on Software Tools for Technology Transfer (STTT)*, 8(6):633–647, 2006.
98. Thomas A. Henzinger. The theory of hybrid automata. In *Symp. on Logic in Computer Science (LICS)*, pages 278–292. IEEE, 1996.
99. Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A user guide to HyTech. In Ed Brinksma, Rance Cleaveland, Kim Gulstrand Larsen, Tiziana Margaria, and Bernhard Steffen, editors, *Intl. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer, 1995.
100. Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What is decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.
101. Thomas A. Henzinger, Peter W. Kopke, and Howard Wong-Toi. The expressive power of clocks. In Zoltán Fülöp and Ferenc Gecseg, editors, *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer, 1995.
102. Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In Werner Kuich, editor, *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 623 of *Lecture Notes in Computer Science*, pages 545–558. Springer, 1992.
103. Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real time systems. *Information and Computation*, 111(2):193–244, 1994.
104. Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In Kim Gulstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 1998.
105. Philippe Herrmann. Timed automata and recognizability. *Information Processing Letters*, 65(6):313–318, 1998.
106. Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002.

107. Hans Hüttel and Kim Gulstrand Larsen. The use of static constructs in a modal process logic. In Albert R. Meyer and Michael A. Taitslin, editors, *Symposium on Logical Foundations of Computer Science*, volume 363 of *Lecture Notes in Computer Science*, pages 163–180. Springer, 1989.
108. Georgeta Igna, Vankatesh Kannan, Yang Yang, Twan Basten, Marc C.W. Geilen, Frits Vaandrager, Marc Voorhoeve, Sebastian de Smet, and Lou J. Somers. Formal modeling and scheduling of datapaths of digital document printers. In Franck Cassez and Claude Jard, editors, *International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume 5215 of *Lecture Notes in Computer Science*, pages 170–187. Springer, 2008.
109. Mohammad Mahdi Jaghoori, Frank S. de Boer, Tom Chothia, and Marjan Sirjani. Schedulability of asynchronous real-time concurrent objects. *Journal of Logic and Algebraic Programming*, 78(5):402–416, 2009.
110. Jan J. Jensen, Jacob Illum Rasmussen, Kim Gulstrand Larsen, and Alexandre David. Guided controller synthesis for climate controller using UPPAAL Tiga. In Jean-François Raskin and P. S. Thiagarajan, editors, *International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume 4763 of *Lecture Notes in Computer Science*, pages 227–240. Springer, 2007.
111. Kenneth Yrke Jørgensen, Kim Gulstrand Larsen, and Jiři Srba. Time-darts: A data structure for verification of closed timed automata. In *Conference on Systems Software Verification (SSV)*, volume 102 of *Electronic Proceedings in Theoretical Computer Science*, pages 141–155, 2012.
112. Jacques Julliand, Hassan Mountassir, and Émilie Oudot. VeSTA: A tool to verify the correct integration of a component in a composite timed system. In Michael J. Butler, Michael G. Hinchey, and María M. Larrondo-Petrie, editors, *International Conference on Formal Engineering Methods (ICFEM)*, volume 4789 of *Lecture Notes in Computer Science*, pages 116–135. Springer, 2007.
113. Miroslav Klimoš, Kim Gulstrand Larsen, Filip Štefaňák, and Jeppe Thaarup. Nash equilibria in concurrent priced games. In Adrian Horia Dediu and Carlos Martín-Vide, editors, *Intl. Conf. on Language and Automata Theory and Applications (LATA)*, volume 7183 of *Lecture Notes in Computer Science*, pages 363–376. Springer, 2012.
114. Sebastian Kupferschmid, Klaus Dräger, Jörg Hoffmann, Bernd Finkbeiner, Henning Dierks, Andreas Podelski, and Gerd Behrmann. Uppaal/DMC – abstraction-based heuristics for directed model checking. In Orna Grumberg and Michael Huth, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *Lecture Notes in Computer Science*, pages 679–682. Springer, 2007.
115. Sebastian Kupferschmid, Jörg Hoffmann, and Kim Gulstrand Larsen. Fast directed model checking via Russian doll abstraction. In C. R. Ramakrishnan and Jakob Rehof, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4963 of *Lecture Notes in Computer Science*, pages 203–217. Springer, 2008.
116. Sebastian Kupferschmid, Martin Wehrle, Bernhard Nebel, and Andreas Podelski. Faster than Uppaal? In Aarti Gupta and Sharad Malik, editors, *Intl. Conf. on Computer-Aided Verification (CAV)*, volume 5123 of *Lecture Notes in Computer Science*, pages 552–555. Springer, 2008.
117. Salvatore La Torre and Margherita Napoli. A decidable dense branching-time temporal logic. In Sanjiv Kapoor and Sanjiva Prasad, editors, *Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 1974 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2000.
118. Salvatore La Torre and Margherita Napoli. Timed tree automata with an application to temporal logic. *Acta Informatica*, 38(2):89–116, 2001.
119. Kim Gulstrand Larsen, Gerd Behrmann, Ed Brinksma, Ansgar Fehnker, Thomas Hune, Paul Pettersson, and Judi Romijn. As cheap as possible: Efficient cost-optimal

reachability for priced timed automata. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Intl. Conf. on Computer-Aided Verification (CAV)*, volume 2102 of *Lecture Notes in Computer Science*, pages 493–505. Springer, 2001.
120. Kim Gulstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: compact data structure and state-space reduction. In *Symposium on Real-Time Systems (RTSS)*, pages 14–24. IEEE, 1997.
121. Kim Gulstrand Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.
122. Kim Gulstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *Intl. Journal on Software Tools for Technology Transfer (STTT)*, 1(1-2):134–152, 1997.
123. Kim Gulstrand Larsen and Jacob Illum Rasmussen. Optimal conditional reachability for multi-priced timed automata. *Theoretical Computer Science*, 390(2-3):197–213, 2008.
124. Sławomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Transactions on Computational Logic*, 9(2), 2008.
125. Denis Lugiez, Peter Niebert, and Sarah Zennou. A partial order semantics approach to the clock explosion problem of timed automata. In Kurt Jensen and Andreas Podelski, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2988 of *Lecture Notes in Computer Science*, pages 296–311. Springer, 2004.
126. Denis Lugiez, Peter Niebert, and Sarah Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theoretical Computer Science*, 345(1):27–59, 2005.
127. Angelika Mader and Hanno Wupper. Timed automaton models for simple programmable logic controllers. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 106–113. IEEE, 1999.
128. Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In Ernst W. Mayr and Claude Puech, editors, *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer, 1995.
129. Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In Nancy Lynch and Bruce H. Krogh, editors, *Int. Workshop on Hybrid Systems: Computation and Control (HSCC)*, volume 1790 of *Lecture Notes in Computer Science*, pages 296–310. Springer, 2000.
130. Robin Milner. *Communication and concurrency*. Prentice Hall Int., 1989.
131. Marius Minea. Partial order reduction for model checking of timed automata. In Jos C. M. Baeten and Sjouke Mauw, editors, *Intl. Conf. on Concurrency Theory (CONCUR)*, volume 1664 of *Lecture Notes in Computer Science*, pages 431–446. Springer, 1999.
132. Jesper B. Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. Difference decision diagrams. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *International Workshop on Computer Science Logic (CSL)*, volume 1862 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 1999.
133. Joël Ouaknine, Alexander Rabinovich, and James Worrell. Time-bounded verification. In Mario Bravetti and Gianluigi Zavattaro, editors, *Intl. Conf. on Concurrency Theory (CONCUR)*, volume 5710 of *Lecture Notes in Computer Science*, pages 496–510. Springer, 2009.
134. Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *Symp. on Logic in Computer Science (LICS)*, pages 54–63. IEEE, 2004.
135. Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In *Symp. on Logic in Computer Science (LICS)*, pages 188–197. IEEE, 2005.
136. Joël Ouaknine and James Worrell. On metric temporal logic and faulty Turing machines. In Luca Aceto and Anna Ingólfsdóttir, editors, *International Conference*

on Foundations of Software Science and Computation Structure (FoSSaCS, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.
137. Joël Ouaknine and James Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logicical Methods in Computer Science*, 3(1), 2007.
138. David Michael Ritchie Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *GI-Conference on Theoretical Computer Science (TCS)*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981.
139. Anuj Puri. Dynamical properties of timed automata. In Anders P. Ravn and Hans Rischel, editors, *Int. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, volume 1486 of *Lecture Notes in Computer Science*, pages 210–227. Springer, 1998.
140. Jean-François Raskin and Pierre-Yves Schobbens. The logic of event-clocks: Decidability, complexity and expressiveness. *Journal of Automata, Languages and Combinatorics*, 4(3):247–286, 1999.
141. Olivier F. Roux and Vlad Rusu. Deciding time-bounded properties for ELECTRE reactive programs with stopwatch automata. In Panos Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II (HSCC)*, volume 999 of *Lecture Notes in Computer Science*, pages 405–416. Springer, 1995.
142. Michał Rutkowski. Two-player reachability-price games on single-clock timed automata. In *Workshop on Quantitative Aspects of Programming Languages (QAPL)*, volume 57 of *Electronic Proceedings in Theoretical Computer Science*, 2011.
143. Ocan Sankur. *Robustness in Timed Automata: Analysis, Synthesis, Implementation*. PhD thesis, Lab. Spécification & Vérification, ENS Cachan, France, 2013.
144. Mathijs Schuts, Yunshan Zhu, Faranak Heidarian, and Frits Vaandrager. Modelling clock synchronization in the Chess gMAC WSN protocol. In Suzana Andova, Annabelle K. McIver, Pedro R. D'Argenio, Pieter J. L. Cuijpers, Jasen Markovski, Carroll C. Morgan, and Manuel Núñez, editors, *Workshop on Quantitative Formal Methods: Theory and Applications (QFM)*, volume 13 of *Electronic Proceedings in Theoretical Computer Science*, pages 41–54, 2009.
145. Josef Tapken and Henning Dierks. MOBY/PLC – graphical development of PLC-automata. In Anders P. Ravn and Hans Rischel, editors, *Int. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, volume 1486 of *Lecture Notes in Computer Science*, pages 311–314. Springer, 1998.
146. Stavros Tripakis. Description and schedulability analysis of the software architecture of an automated vehicle control system. In Alberto L. Sangiovani-Vincentelli and Joseph Sifakis, editors, *Int. Conf. on Embedded Software (EMSOFT)*, volume 2491 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2002.
147. Stavros Tripakis. Checking timed Büchi automata emptiness on simulation graphs. *ACM Transactions on Computational Logic*, 10(3), 2009.
148. Stavros Tripakis and Karine Altisen. On-the-fly controller synthesis for discrete and dense-time systems. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *World Congress on Formal Methods (FM)*, volume 1708 of *Lecture Notes in Computer Science*, pages 233–252. Springer, 1999.
149. Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design (FMSD)*, 18(1):25–68, 2001.
150. Stavros Tripakis and Sergio Yovine. Timing analysis and code generation of vehicle control software using Taxys. In Klaus Havelund and Grigore Roşu, editors, *International Workshop on Runtime Verification (RV)*, volume 55 of *Electronic Notes in Theoretical Computer Science*, pages 277–286. Elsevier Science, 2001.
151. Stavros Tripakis, Sergio Yovine, and Ahmed Bouajjani. Checking timed Büchi automata emptiness Efficiently. *Formal Methods in System Design (FMSD)*, 26(3):267–292, 2005.
152. Farn Wang. Model-checking distributed real-time systems with states, events, and multiple fairness assumptions. In Charles Rattray, Savi Maharaj, and Carron Shank-

land, editors, *International Conference on Algebraic Methodology and Software Technology (AMAST)*, volume 3116 of *Lecture Notes in Computer Science*, pages 553–567. Springer, 2004.
153. Farn Wang. Efficient model-checking of dense-time systems with time-convexity analysis. *Theoretical Computer Science*, 467:89–108, 2013.
154. Farn Wang, Geng-Dian Huang, and Fang Yu. TCTL inevitability analysis of dense-time systems: From theory to engineering. *IEEE Transaction on Software Engineering*, 32(7):510–526, 2006.
155. Farn Wang, Li-Wei Yao, and Ya-Lan Yang. Efficient verification of distributed real-time systems with broadcasting behaviors. *Real-Time Systems*, 47(4):285–318, 2011.
156. Libor Waszniowski and Zdeněk Hanzálek. Formal verification of multitasking applications based on timed automata model. *Real-Time Systems*, 38(1):39–65, 2008.
157. Wang Yi, Paul Pettersson, and Mats Daniels. Automatic verification of real-time communicating systems by constraint-solving. In Dieter Hogrefe and Stefan Leue, editors, *Intl. Conf. on Formal Description Techniques (FORTE)*, volume 6 of *IFIP Conference Proceedings*, pages 243–258. Chapman & Hall, 1995.
158. Sergio Yovine. Kronos: A verification tool for real-time systems. *Intl. Journal on Software Tools for Technology Transfer (STTT)*, 1(1-2):123–133, 1997.
159. Sarah Zennou, Manuel Yguel, and Peter Niebert. ELSE: A new symbolic state generator for timed automata. In Kim Gulstrand Larsen and Peter Niebert, editors, *Int. Workshop on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume 2791 of *Lecture Notes in Computer Science*, pages 273–280. Springer, 2003.

Index

automaton
  timed, 4

bisimulation, 11, 38
  time-abstracted, 8

clock, 3
clock constraint, 3
corner-point abstraction, 31

difference bound matrix (DBM), 26

energy constraints, 34

language
  of a timed automaton, 13

region, 8
region automaton, 8
region equivalence, 7

simulation, 11

temporal logic
  MTL, 19
  TCTL, 22
  TPTL, 19
  weighted CTL, 33
timed automaton, 4
  weighted, 30
timed game, 36
timed word, 3

zone, 25